

**Document Title:** IoT/Time series data and real-time streamings with NiFi

**Date:** September 2019

**Version:** 8.0

**Notes:** pre-created OneNode-cluster

**Author:** Filippo Lambiente, André Molenaar

## **IoT/Time series data and real-time streaming with NiFi**

### **Table of Contents**

Objectives	2
Labs that we will go through	2
Lab 1 – User Registration for Cloudera on Azure hands on Labs	3
Lab 2 - Access the pre-provisioned Cloudera cluster	7
Lab 3: Get the Connected Things/Data Sources	15
generateDataSensors flow	16
Lab 4: Define the Kudu tables for the Connected Thing time series data	39
Lab 5: Create the real-time ingest pipeline	42
ingestDataSensors flow	42
Lab 6: Run and verify the real-time ingest pipelines	75
Hint	92
Lab 7: Improve Kudu table partitioning strategies	94
Partition strategy improvement #1	94
Partition strategy improvement #2	98
Simple Queries	99
APPENDIX - NiFi solution	103
APPENDIX - Promote a flow in Production	103
APPENDIX - Using NiFi to ingest Web logs into EDH	108
Getting the Web Logs file	108
Creating a Web Logs table	113
Creating the NiFi Flow	115
Run the Flow	157

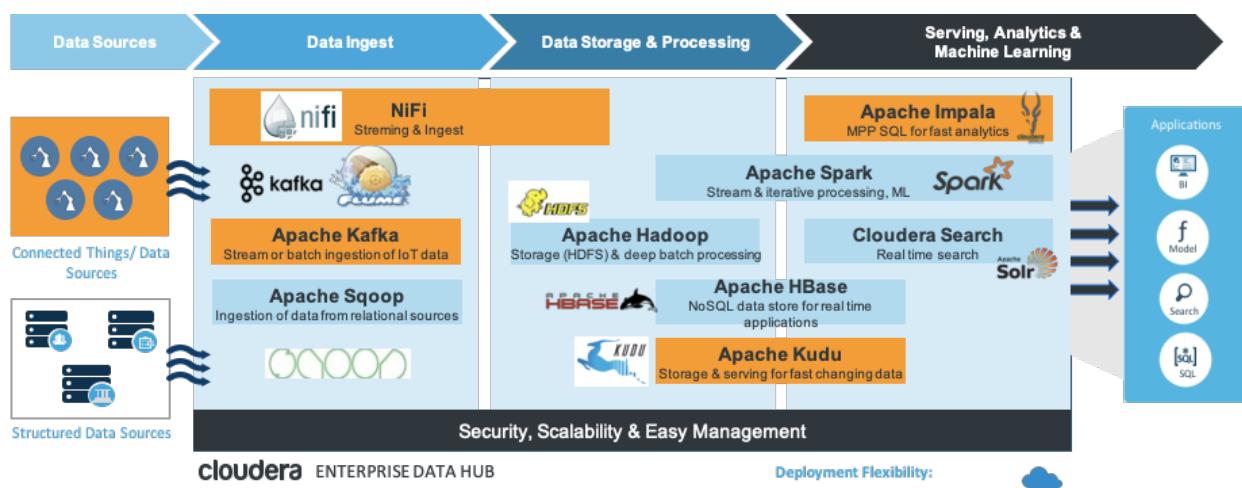
# Objectives

The goal of this track is to drive you through the implementation of an end-to-end scenario for real-time, time series, data ingestion with the Cloudera platform.

In particular:

- to create a simple & quick end-to-end demo in the IoT context from Data Ingest, to Data Storage, Processing and Serving;
- Eliminate the need of writing code
- leveraging the components of the platform and
- keeping the moving parts to a minimum

The picture below depicts the architecture of this demo scenario:



We highlight in orange the components we're going to use

## Labs that we will go through

Here is the list of the labs that will need to do in order to achieve our goals:

1. Get a pre-provisioned Cloudera cluster from Azure Marketplace
2. Preparing the cluster for real-time streaming ingestion
3. Get the Connected Things/Data Sources
4. Define the Kudu tables for the Connected Thing time series data
5. Create the real-time pipeline that reads data from the Connected Things and insert them into Kudu
6. Run and verify the real-time ingest pipeline
7. Improve Kudu table partitioning strategies and adjust the ingest pipeline
8. Versioning
9. Closing discussion

# Lab 1 – User Registration for Cloudera on Azure hands on Labs

Go to the following link <http://bit.ly/2Ufc6JN> for the registration page

The screenshot shows a registration form for a Microsoft Cloud Workshop. The title is "Microsoft Cloud Workshops". Below it, the specific lab is identified as "Lab with NiFi/Kafka/Kudu on Azure" and the date is "cloudera - 04/05/2019". The organizer is listed as "By : Cloudera". A note says "Please sign up to get access to the environment." To the right, there's a "Register Now" section with fields for "First Name\*", "Last Name\*", "Work email\*", "Organization\*", "Country \*", and "Activation Code \*". Below these fields is a privacy statement from Cloudera. At the bottom right is a blue "SUBMIT" button.

Fill in the fields and click **SUBMIT**

Your instructor will give you the **Activation Code**

Only corporate emails can be used to register (no gmail, yahoo, outlook or hotmail)

This screenshot shows the same registration page after submission. The "Thank you for registering!" message is displayed prominently. Below it, a note says "You'll receive an email with further instructions." The rest of the page content is identical to the first screenshot, including the lab title, date, organizer, and registration fields.

Check the email:

Invite for On demand lab - Lab with NiFi/Kafka/Kudu on Azure - 04/05/2019 Inbox × 🖨️

No Reply (CloudLabs) <noreply@cloudlabs.ai>1:48 PM (1 minute ago) ★ ↶

to me ▾

Lab with NiFi/Kafka/Kudu on Azure - 04/05/2019

Dear Filippo Lambiente

You are invited to take an On demand lab - **Lab with NiFi/Kafka/Kudu on Azure - 04/05/2019**.

Please note that the maximum duration is 3 hours from the start of the lab, after which it will be automatically deallocated. This invite will expire on Saturday, April 6, 2019.

[Launch Lab](#)

Thank you and have a great On Demand Lab!

Then click on **Launch Lab** button

Microsoft Cloud Workshops

cloudera By : Cloudera

**Lab with NiFi/Kafka/Kudu on Azure - 04/05/2019**

Overview

**Lab**

⌚ 3 hours

**LAUNCH LAB**

Azure,Cloudera

Click again on the **LAUNCH LAB** button and you should get your Azure credentials and cluster details:

# Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

✓ Your On Demand Lab is ready (3 hour(s), 57 minute(s) remaining)

**Environment Details**      **Virtual Machines**

**Azure Credentials**

Here are your credentials to login to Microsoft Azure and access the On Demand Lab

Username	odl_user_59475@clouderahol.onmicrosoft.com	
Password	qojz78NDJ*H5	

**Environment Details**

**Resource Group : ODL-cdh-59475**

CLOUDERA MANAGER URL	cl59475-mn0.southcentralus.cloudapp.azure.com:7180	
ADMIN USERNAME	cloudera	
ADMIN PASSWORD	Cloudera_123	
CLOUDERA MANAGER USERNAME	cloudera	
CLOUDERA MANAGER PASSWORD	Cloudera_123	

Lab Guide : <https://cloudera.box.com/s/9ib8iixldxq4md6xfeu7iausqb3yyjk>

Use your temporary Azure credentials to log into Azure at the following URL =  
<https://azure.microsoft.com/en-us/>

**Azure Credentials**

Here are your credentials to login to Microsoft Azure and access the On Demand Lab

Username	odl_user_59475@clouderahol.onmicrosoft.com	
Password	oxxy77SZV*TK	

Click on **Sing-in** and provide the Azure credentials and once logged-in you will see the Azure in the top right hand corner Portal and ODL\_User

Click on **Portal**



# Welcome to Microsoft Azure

Let's show you around before you get started.

**Start tour**

**Maybe later**

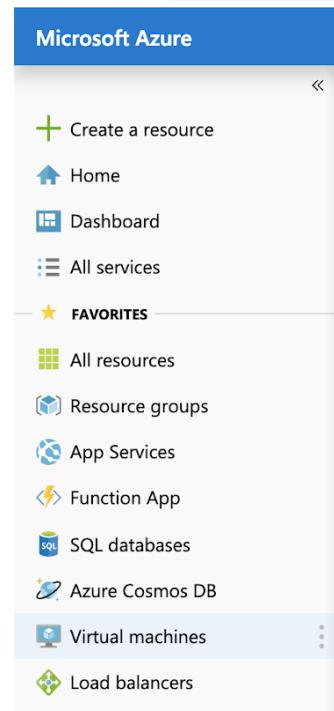
Click **Maybe later** button.

## Lab 2 - Access the pre-provisioned Cloudera cluster

The details of the preprovisioned cluster are available here:

<u>Environment Details</u>	
<b>Resource Group : ODL-cdh-59475</b>	
CLOUDERA MANAGER URL	cl59475-mn0.southcentralus.cloudapp.azure.com:7180
ADMIN USERNAME	cloudera
ADMIN PASSWORD	Cloudera_123
CLOUDERA MANAGER USERNAME	cloudera
CLOUDERA MANAGER PASSWORD	Cloudera_123

In the Azure portal check the VMs created by clicking on the **Virtual machines** link



This is your pre-provisioned cluster with a single-node cluster:

# Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

Home > Virtual machines

## Virtual machines

CloudLabs AI (Spektra)

Docu

+ Add Reservations Edit columns Refresh Assign tags Start Restart Stop Delete Services

**Subscriptions:** Microsoft Managed Labs Spektra - 17

Filter by name... All resource groups All types All locations All tags No grouping

1 items

<input type="checkbox"/> NAME ↑↓	TYPE ↑↓	STATUS	RESOURCE GROUP ↑↓	LOCATION ↑↓	SOURCE	MAINTENANCE STATUS	SUBSCRIPTI
<input type="checkbox"/>  cdhvms	Virtual machine	Running	cdf-83379	East US	Shared image	-	Microso

## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

Click on the Virtual machine name (cdhvm) to see its public ip address as well as its private ip address and its dns name:

The screenshot shows the Azure portal interface for a virtual machine named 'cdhvm'. The left sidebar has options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Networking, and Disks. The main pane displays the VM details. A blue bar at the top says 'Advisor (1 of 3): Enable virtual machine backup to protect your data from corruption and accidental deletion'. Below it, the VM details are listed:

Resource group (change)	:	cdf-83379
Status	:	Running
Location	:	East US
Subscription (change)	:	Microsoft Managed Labs Spektra - 17
Subscription ID	:	ea8323fb-dd33-48d1-8ffb-3d771bc5f05c
Computer name	:	cdhvm:jgcmmgxg0rpefjbibziadj2dza.bx.internal.cloudapp.net
Operating system	:	Linux (centos 7.6.1810)
Size	:	Standard DS4 v2 (8 vcpus, 28 GiB memory)
Ephemeral OS disk	:	N/A
Public IP address	:	40.76.204.159
Private IP address	:	10.0.0.4
Virtual network/subnet	:	vNet/default
DNS name	:	cdhvmti2hewrcsr2o.eastus.cloudapp.azure.com

Write down the ip address (or the dns name) for later reference in the workshop.

Check of all services of your Cloudera Data Hub cluster are up-and-running.  
Navigate to the Cloudera Manager UI using the url (change the ip address in the provided example with your ip address):  
<http://40.76.204.159:7180>

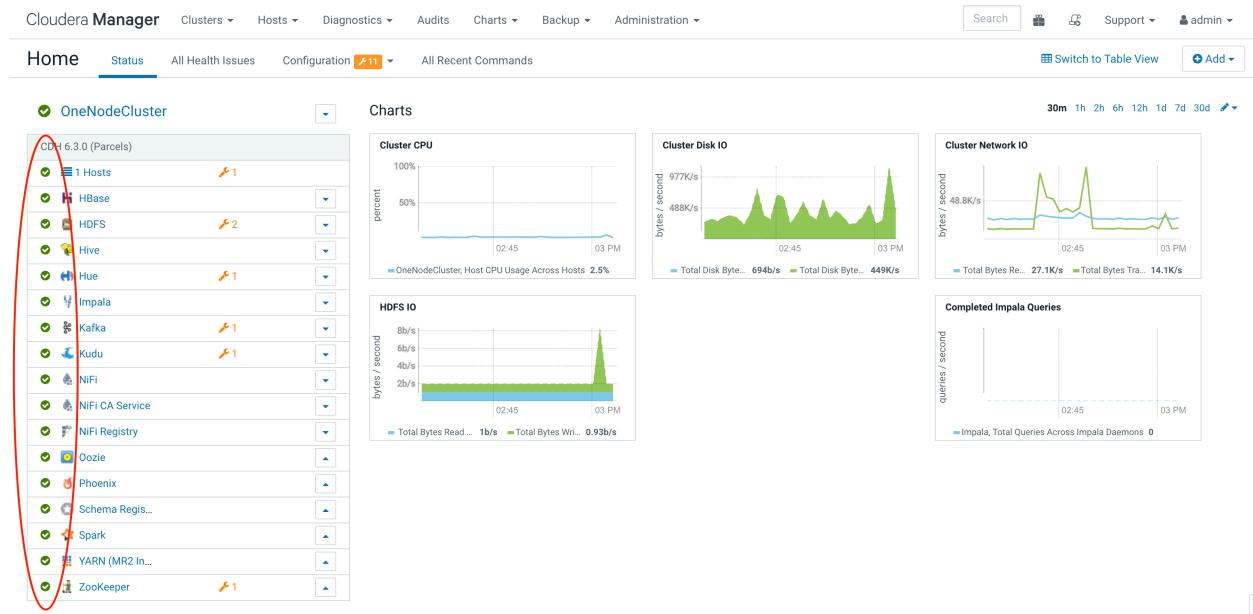
The link will take you to the Cloudera Manager login page:

The screenshot shows the Cloudera Manager login page. It features a simple form with the following fields:

- A text input field containing 'admin'.
- A text input field containing '.....' (the password).
- A checkbox labeled 'Remember me'.
- A large blue 'Sign In' button.

Login to Cloudera Manager using user 'admin' with the password 'admin'.

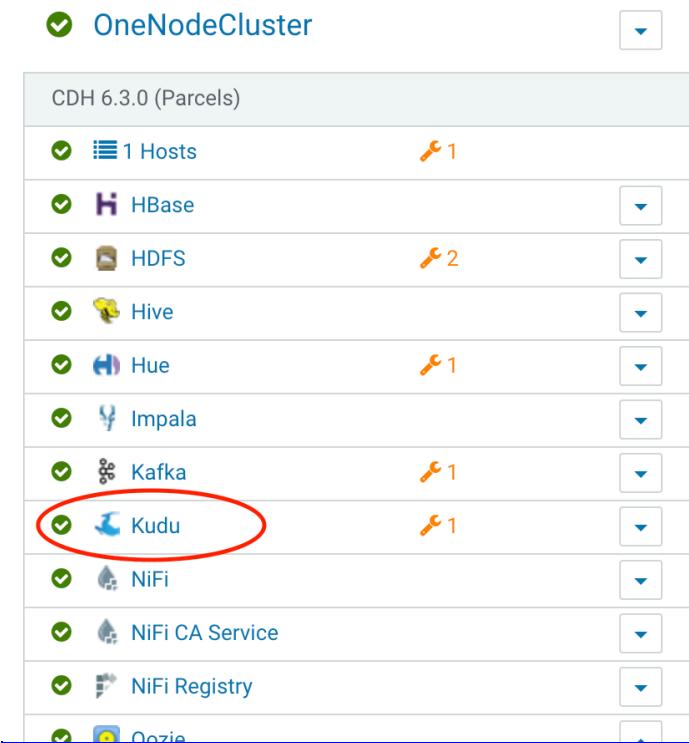
After login you will see all services running on the cluster. If all services are ‘green’, you are ready to proceed. If some are red, you may need to wait a bit longer before you can proceed.



Change the replication factor in Kudu:

In this lab, we will write data to Kudu. By default, data in kudu will be replicated 3 times on different tablet servers, to support High Availability. But since this test-cluster only has 1 node, 3 times replication cannot be used. By switching back to 1 replica, we will be able to use kudu in this workshop. Change the replication factor as follows:

Go to your Cloudera Manager interface, and select the Kudu service:



The screenshot shows the Cloudera Manager interface with the cluster name "OneNodeCluster" selected. The service list includes:

- CDH 6.3.0 (Parcels)
- 1 Hosts (1)
- HBase
- HDFS (2)
- Hive
- Hue (1)
- Impala
- Kafka (1)
- Kudu** (1)
- NiFi
- NiFi CA Service
- NiFi Registry
- Oozie

Go to the 'Configuration' page, search for 'replic', and change the number of replica's to 1. Hit 'Save Changes' to commit your change.

# Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

The screenshot shows the Cloudera Manager interface for the Kudu service. The top navigation bar includes 'Clusters', 'Hosts', 'Diagnostics', 'Audits', 'Charts', 'Backup', and 'Administration'. The 'Clusters' tab is selected. Below it, 'OneNodeCluster' and 'Kudu' are listed with an 'Actions' dropdown. The main content area has tabs for 'Status', 'Instances', 'Configuration' (which is highlighted with a red circle), 'Commands', 'Charts Library', 'Audits', 'Kudu Master Web UI' (with a blue link icon), and 'Quick Links'. A search bar at the top right contains the term 'replic' (also circled in red). On the left, a 'Filters' sidebar shows counts for 'SCOPE' (Kudu (Service-Wide) 2, Master 1, Tablet Server 1), 'CATEGORY' (Advanced 0, Logs 0, Main 4, Monitoring 0, Performance 0, Ports and Addresses 0, Resource Management 0, Security 0), and 'STATUS' (Error 0, Warning 0, Edited 1, Non-default 1, Has Overrides 0). The main configuration table lists three items: 'Maximum Number of Per-tablet-server Replica Moves' (rb\_max\_moves\_per\_server, Kudu (Service-Wide), value 5), 'Maximum Allowed Duration Without Rebalancer Progress' (rb\_max\_staleness\_interval\_sec, Kudu (Service-Wide), value 300), and 'Default Number of Replicas' (default\_num\_replicas, Master Default Group, value 1, also circled in red). A checkbox for 'Tablet Server Default Group' is present but unchecked. At the bottom, there's a note '1 Edited Value' and a 'Reason for change...' input field, followed by a large blue 'Save Changes' button (also circled in red).

After the change, the service needs to be restarted.

Navigate to the Cloudera Manager homepage, and click on the restart service button behind the Kudu service:

Cloudera Manager

Clusters ▾ Hosts ▾ Diagnos

Home Status All Health Issues Configuration

OneNodeCluster

CDH 6.3.0 (Parcels)

1 Hosts	1
HBase	
HDFS	2
Hive	
Hue	1
Impala	
Kafka	1
Kudu	2
NiFi	
NiFi CA Service	

Hit the 'Restart Stale Services' button.

# Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

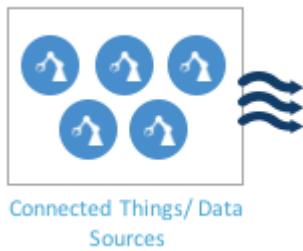
The screenshot shows the Cloudera Manager interface under the 'Clusters' tab. A search bar at the top right contains the text 'kudu'. Below it, a 'Support' button and a user account dropdown for 'admin' are visible. The main content area is titled 'OneNodeCluster' and 'Stale Configurations'. On the left, a 'Filters' sidebar shows two entries: 'FILE' (File: gflagfile) and 'SERVICE' (Kudu). The 'FILE' entry has a count of 1. The 'SERVICE' entry also has a count of 1. The main pane displays a configuration file named 'gflagfile' with five lines of code. The fifth line, '-log\_force\_fsync\_all=false', is highlighted with a red oval. At the bottom right of the main pane is a blue button labeled 'Restart Stale Services'.

Select the ‘Re-deploy client configuration’ checkbox, and hit ‘Restart Now’ button.

The screenshot shows a confirmation dialog titled 'Restart Stale Services'. On the left, there are two buttons: 'Review Changes' (highlighted with a blue circle) and 'Command Details'. The main body of the dialog is titled 'Review Changes' and contains the text: 'All services running with outdated configurations in the cluster and their dependencies will be restarted.' Below this text is a checkbox labeled 'Re-deploy client configuration', which is checked and highlighted with a red oval. At the bottom right of the dialog are two buttons: 'Back' and 'Restart Now' (highlighted with a red oval).

Wait till the restart of the Kudu service is completed and all services on the cluster are running again.

## Lab 3: Get the Connected Things/Data Sources



For the purpose of this demo, for sake of simplicity, we will simulate the remote, connected things. Here you will see how to create a NiFi pipeline in order to simulate generic remote Things that generate time series data.

The remote, connected Things that we are going to simulate are sensors defined by the following 3 fields:

- Sensor ID
- Sensor Value
- Timestamp

Where the Sensor ID is the unique identifier of the sensor, and for our demo we can assume that there are 10.000 sensors; value and timestamp are the timeseries reading.

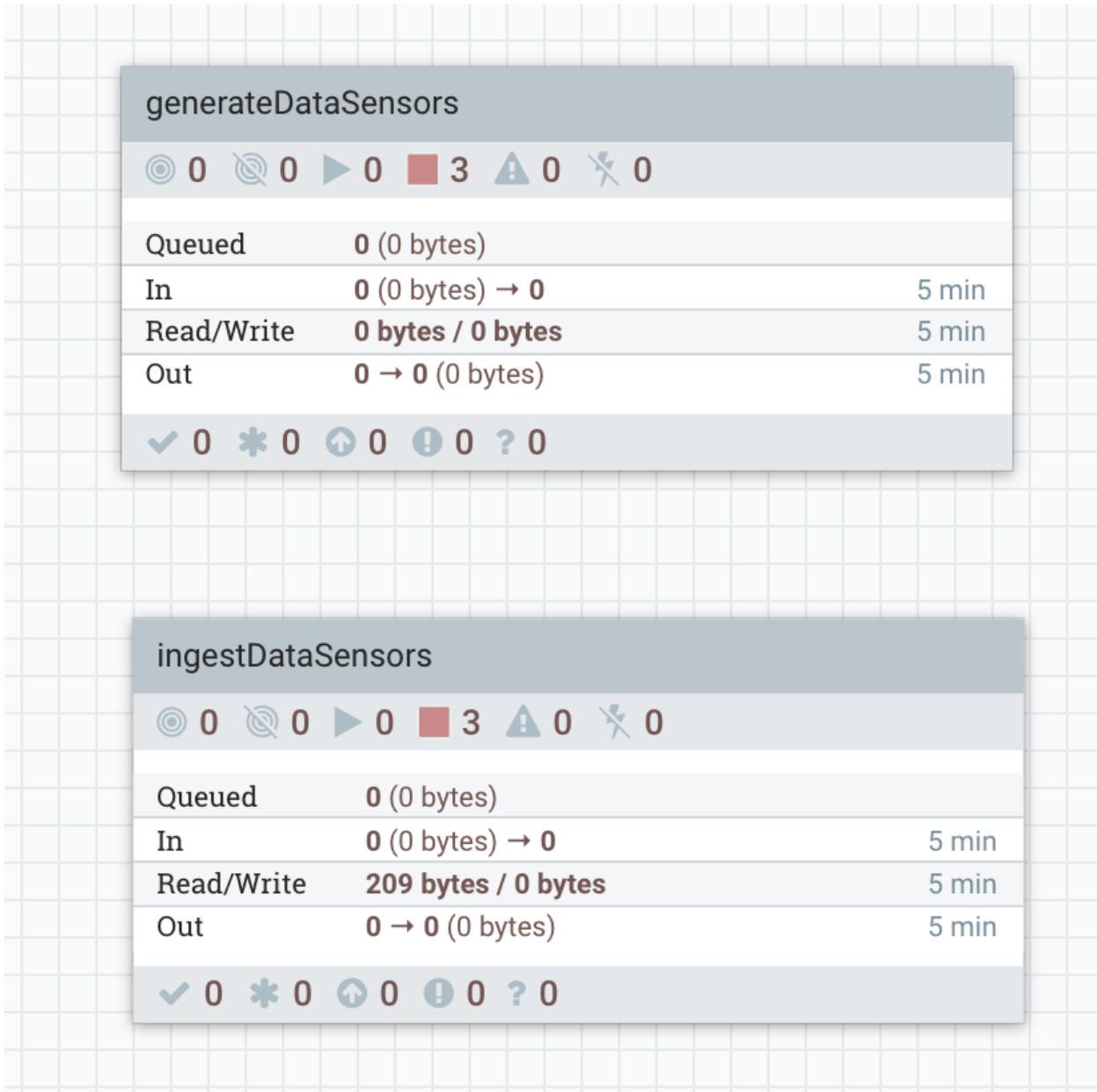
You are free to model a more complex remote Things, but for the rest of this exercise we will consider the above 3 fields for the remote Thing.

Once that we have conceptually modelled the remote Thing, now we are going to build a pipeline in NiFi that will act as a simulator for those 10.000 remote Things; it generates the random sensor reading and publishes the readings to a Kafka topic.

With NiFi we are going to create two flows with these goals:

- One flow that will generate the time-series sensor reading, we are going to call *generateDataSensors*
- A second flow that will read the time-series sensor reading and store into Kudu, we are going to call *ingestDataSensors*

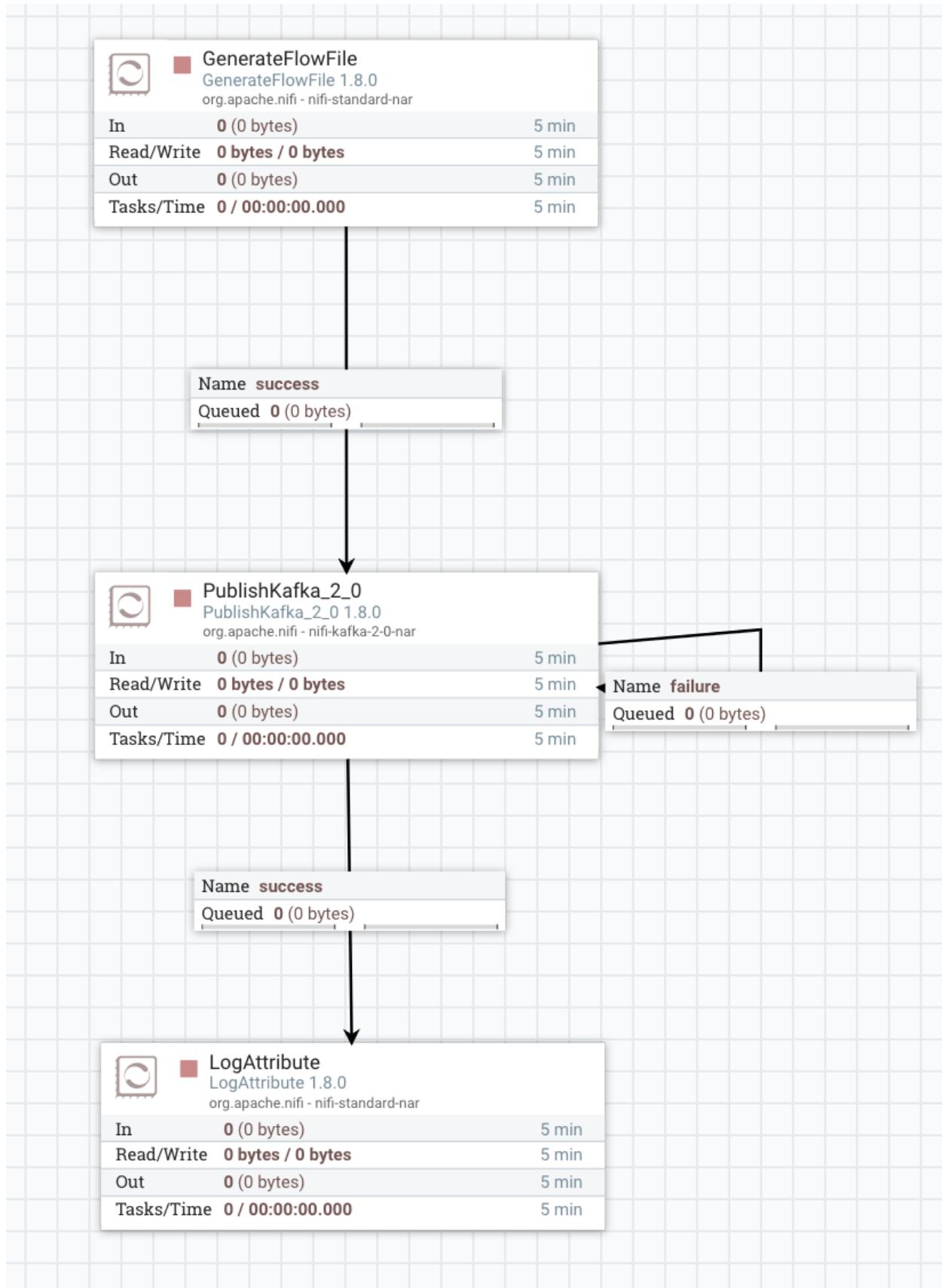
At the high level we will have the following:



Let's start with the first flow.

generateDataSensors flow

This flow is going to look like the following picture

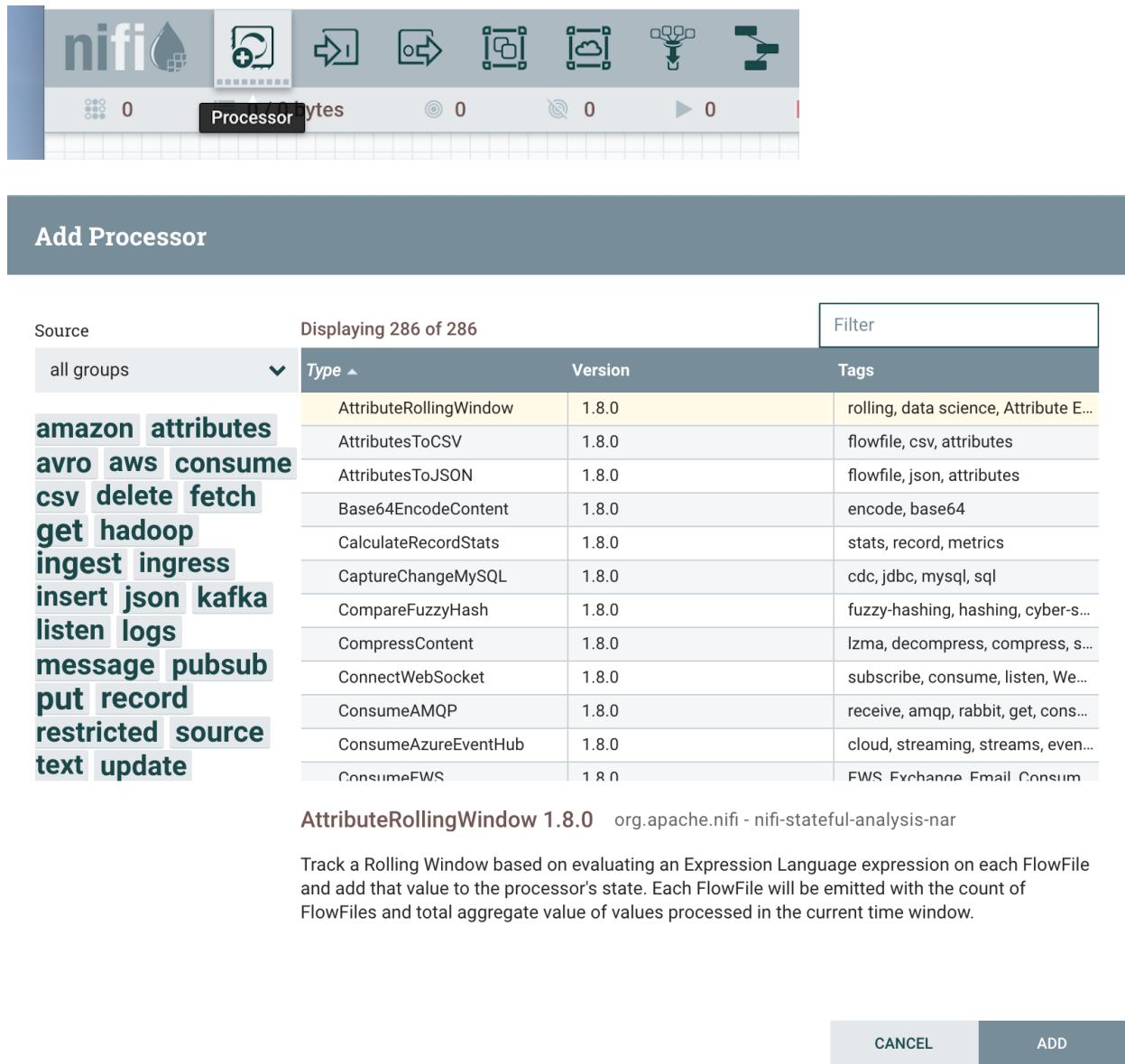


## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

Follow these steps to create the above flow in your NiFi instance.

Login to the NiFi web UI (use your ip address instead of this example)  
<http://40.76.204.159:8080/nifi/>

On the top-left corner, click on the **Processor** icon and drag it into the canvas:



The screenshot shows the NiFi web interface. At the top, there's a toolbar with icons for various processor types: Source, Sink, Transform, Logic, and Connectors. Below the toolbar, a counter for 'Processor' shows '0'. The main area is titled 'Add Processor' and displays a list of 286 available processors. The list is filtered by 'Type' (sorted by name) and includes columns for 'Source', 'Type', 'Version', and 'Tags'. One processor, 'AttributeRollingWindow', is highlighted in yellow. Below the list, its detailed description is shown: 'AttributeRollingWindow 1.8.0 org.apache.nifi - nifi-stateful-analysis-nar'. It is described as tracking a Rolling Window based on an Expression Language expression. At the bottom right of the dialog are 'CANCEL' and 'ADD' buttons.

Source	Type	Version	Tags
all groups	AttributeRollingWindow	1.8.0	rolling, data science, Attribute E...
	AttributesToCSV	1.8.0	flowfile, csv, attributes
	AttributesToJson	1.8.0	flowfile, json, attributes
	Base64EncodeContent	1.8.0	encode, base64
	CalculateRecordStats	1.8.0	stats, record, metrics
	CaptureChangeMySQL	1.8.0	cdc, jdbc, mysql, sql
	CompareFuzzyHash	1.8.0	fuzzy-hashing, hashing, cyber-s...
	CompressContent	1.8.0	lzma, decompress, compress, s...
	ConnectWebSocket	1.8.0	subscribe, consume, listen, We...
	ConsumeAMQP	1.8.0	receive, amqp, rabbit, get, cons...
	ConsumeAzureEventHub	1.8.0	cloud, streaming, streams, even...
	ConsumeEWS	1.8.0	EWS Exchange Email Consum...

In the **Filter** field type *generate* and select **GenerateFlowFile** and click **ADD**

Add Processor

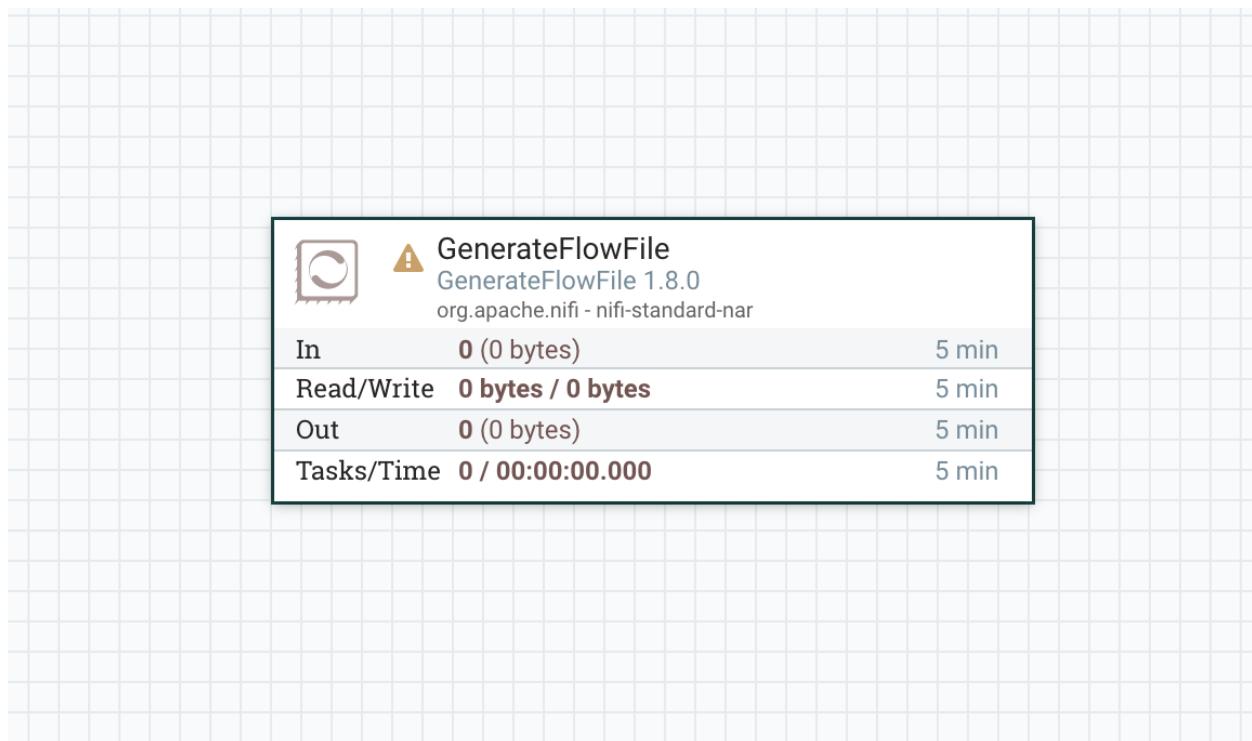
Source	Type	Version	Tags
all groups	▼	Type ▲	generate
amazon attributes	GenerateFlowFile	1.8.0	random, test, generate
avro aws consume	GenerateTableFetch	1.8.0	database, select, query, fetch, jd...
csv delete fetch			
get hadoop			
ingest ingress			
insert json kafka			
listen logs			
message pubsub			
put record			
restricted source			
text update			

GenerateFlowFile 1.8.0 org.apache.nifi - nifi-standard-nar

This processor creates FlowFiles with random data or custom content. GenerateFlowFile is useful for load testing, configuration, and simulation.

CANCEL ADD

As a result you will get in the canvas the following component:



We are going to use this component in order to generate random time-series sensor reading.  
Double click on this component:

## Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Name: GenerateFlowFile     Enabled    Automatically Terminate Relationships ?  
 success

Id: 60c0561d-2ffb-3486-8b4b-44278a501545

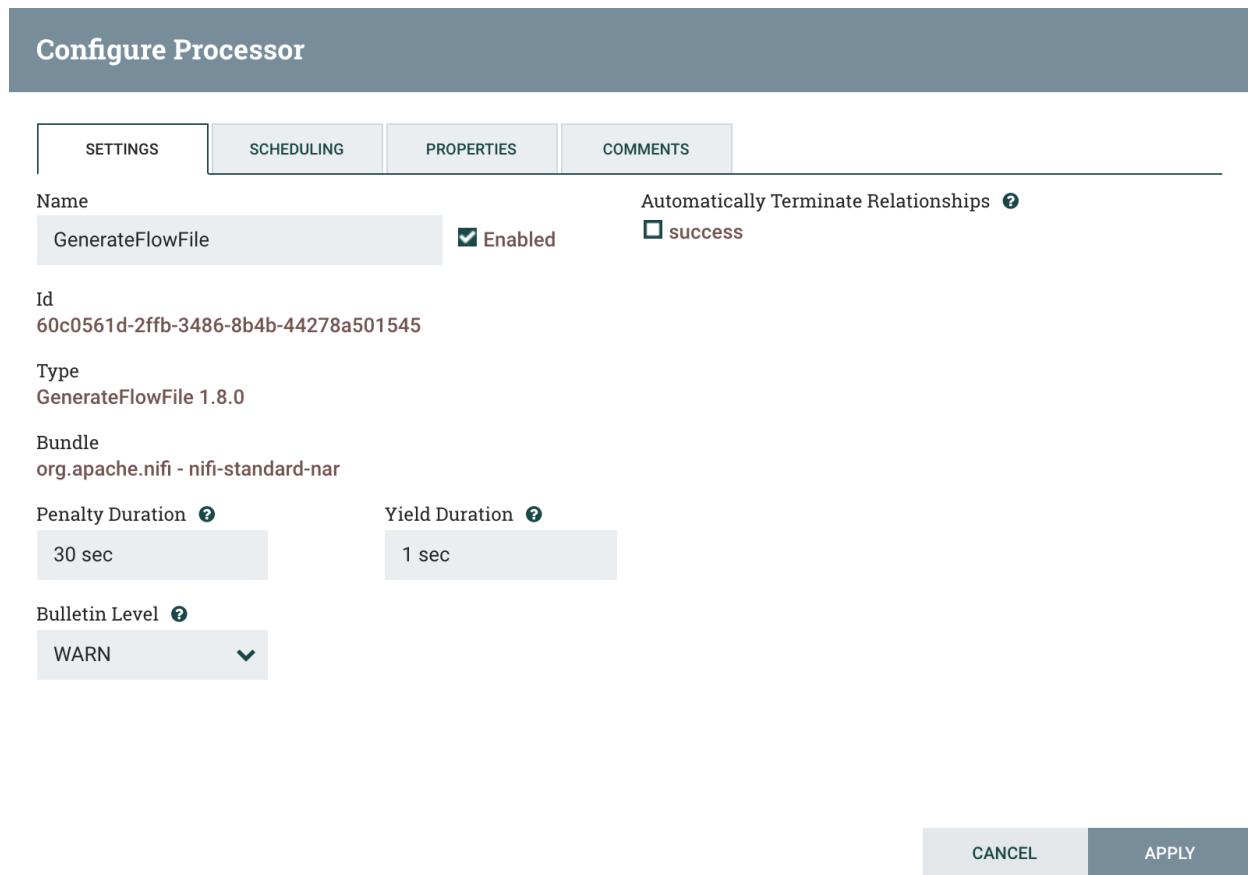
Type: GenerateFlowFile 1.8.0

Bundle: org.apache.nifi - nifi-standard-nar

Penalty Duration ?: 30 sec    Yield Duration ?: 1 sec

Bulletin Level ?:

CANCEL    APPLY



Go to the **SCHEDULING** tab:

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Scheduling Strategy ?  
Timer driven

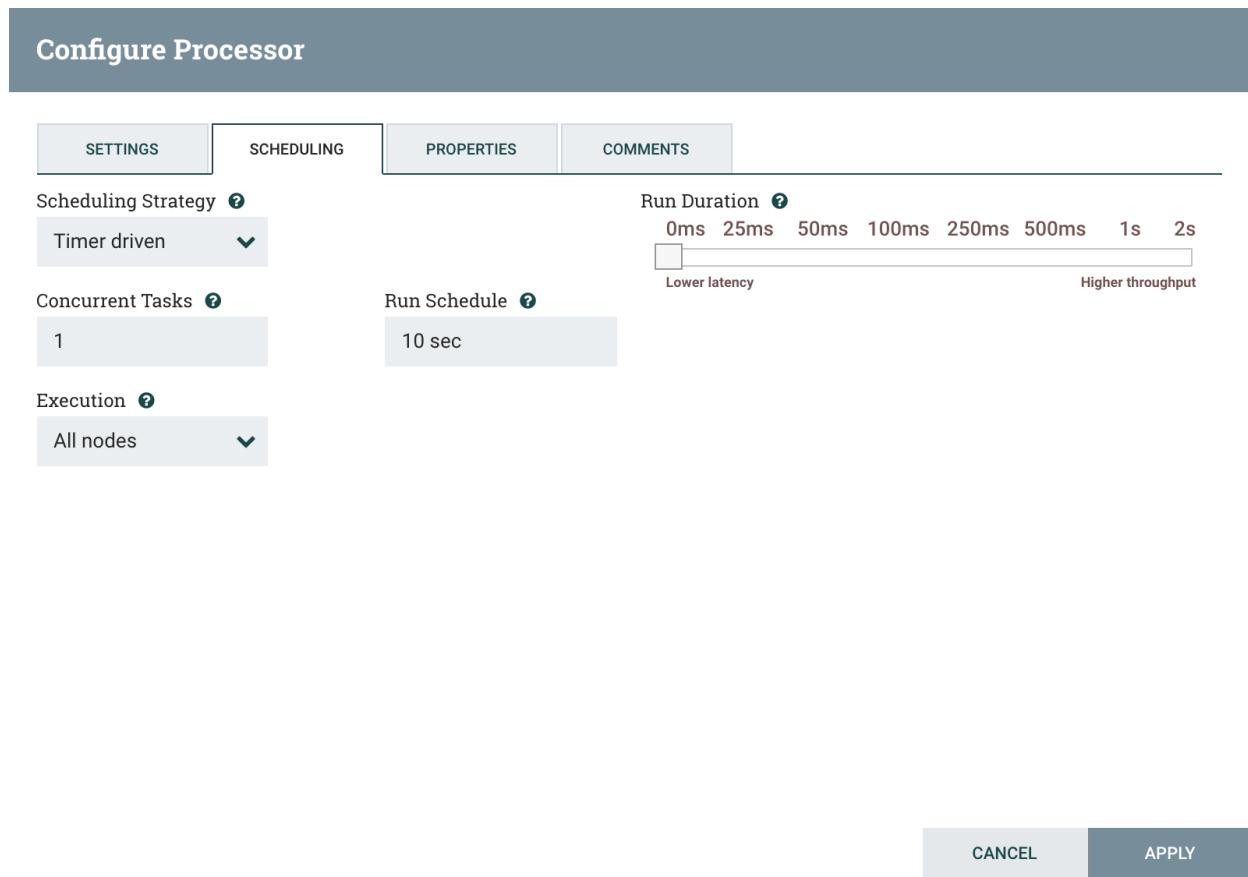
Concurrent Tasks ?  
1

Execution ?  
All nodes

Run Duration ?  
0ms 25ms 50ms 100ms 250ms 500ms 1s 2s  
Lower latency Higher throughput

Run Schedule ?  
10 sec

CANCEL APPLY



And change the **Run Schedule** to 10 sec, in order to generate a sensor reading every 10 seconds.  
Then go to the **PROPERTIES** tab:

The screenshot shows the 'Configure Processor' interface for a specific processor. At the top, there are tabs for 'SETTINGS', 'SCHEDULING', 'PROPERTIES', and 'COMMENTS'. The 'PROPERTIES' tab is selected. Below the tabs, a section titled 'Required field' contains a table with the following data:

Property	Value
File Size	0B
Batch Size	1
Data Format	Text
Unique FlowFiles	false
Custom Text	{"sensorID":\${random():mod(10000):plus(1)},"sensorValue":\$...}
Character Set	UTF-8

In the bottom right corner of the properties table, there is a small '+' icon. At the very bottom of the screen, there are 'CANCEL' and 'APPLY' buttons.

And in the **Value** of the **Custom Text** field add the following:

```
{
    "sensorID":${random():mod(10000):plus(1)},
    "sensorValue":${random():mod(255):plus(1)},
    "sensorTimestamp":${now():toNumber():multiply(1000)}
}
```

This expression will generate JSON messages like:

```
{
    "sensorID":5172.0,
    "sensorValue":16.0,
    "sensorTimestamp":1.547053587308E12
}
```

Basically we are using the NiFi Expression Language to generate sensor reading with the following characteristics:

- A sensor identifier with a value from 1 to 10000
- A sensor reading with a value from 1 to 255
- And a timestamp

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
File Size	0B
Batch Size	1
Data Format	Text

Unique FlowFiles

Custom Text

Character Set

```
1 {
2 "sensorID":${random():mod(10000):plus(1
3 "sensorValue":${random():mod(255):plus(
4 "sensorTimestamp":${now():toNumber():mu
5 }
6 }
```

Set empty string

CANCEL OK

CANCEL APPLY

And click **OK** and then click **APPLY**:

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
File Size	0B
Batch Size	1
Data Format	Text
Unique FlowFiles	false
Custom Text	{"sensorID":\${random():mod(10000):plus(1)},"sensorValue":\$...}
Character Set	UTF-8

CANCEL APPLY

Now that we have a component that generates random reading, we can proceed with our flow and add the component that will be responsible to publish the message to a Kafka topic.

Drag & Drop again the **Processor** icon into the canvas, and select PublishKafka\_2\_0

Add Processor

Source	Type	Version	Tags
all groups	PublishKafkaRecord_0_11	1.8.0	PubSub, Message, csv, Kafka, j...
	PublishKafkaRecord_1_0	1.8.0	PubSub, 1.0, Message, csv, Kaf...
	PublishKafkaRecord_2_0	1.8.0	PubSub, Message, 2.0, csv, Kaf...
	PublishKafka_0_10	1.8.0	0.10.x, PubSub, Message, Kaf...
	PublishKafka_0_11	1.8.0	PubSub, Message, Kafka, Apac...
	PublishKafka_1_0	1.8.0	PubSub, 1.0, Message, Kafka, A...
	PublishKafka_2_0	1.8.0	PubSub, Message, 2.0, Kafka, A...
	PublishMQTT	1.8.0	MQTT, publish, IOT
	PutCloudWatchMetric	1.8.0	amazon, publish, cloudwatch, ...
	PutSNS	1.8.0	amazon, publish, sns, topic, aw...
	PutSQS	1.8.0	SQS, Amazon, AWS, Queue, Put,...

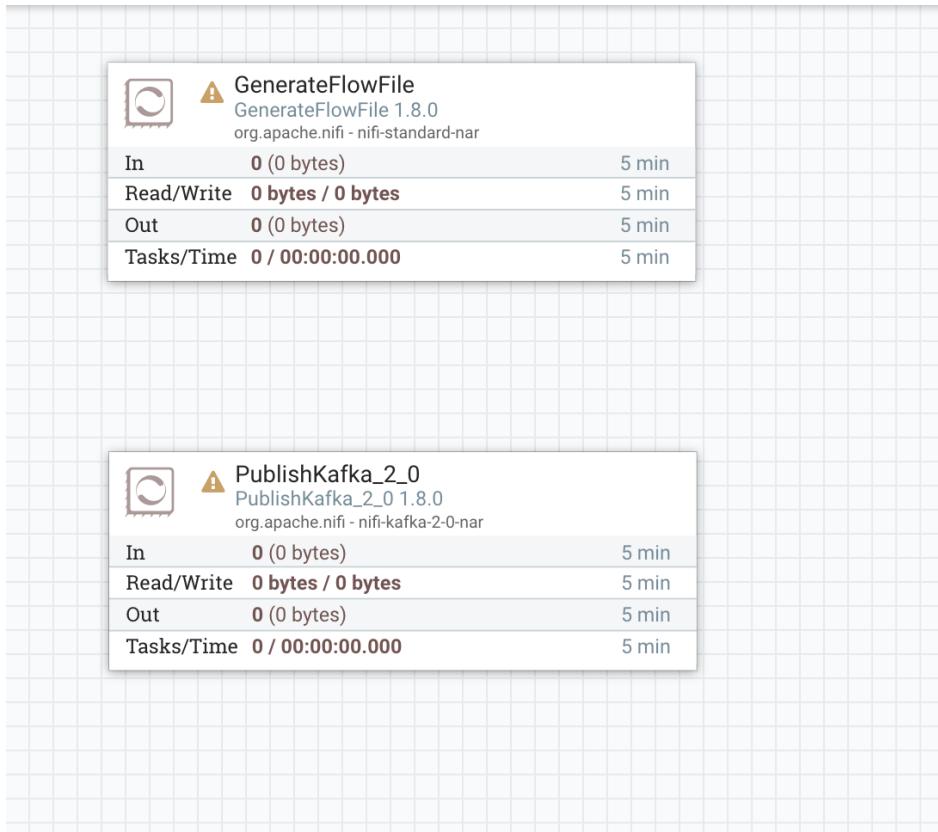
**PublishKafka\_2\_0 1.8.0 org.apache.nifi - nifi-kafka-2-0-nar**

Sends the contents of a FlowFile as a message to Apache Kafka using the Kafka 2.0 Producer API. The messages to send may be individual FlowFiles or may be delimited, using a user-specified delimiter, such as a new-line. The complementary NiFi processor for fetching messages is ConsumeKafka\_2\_0.

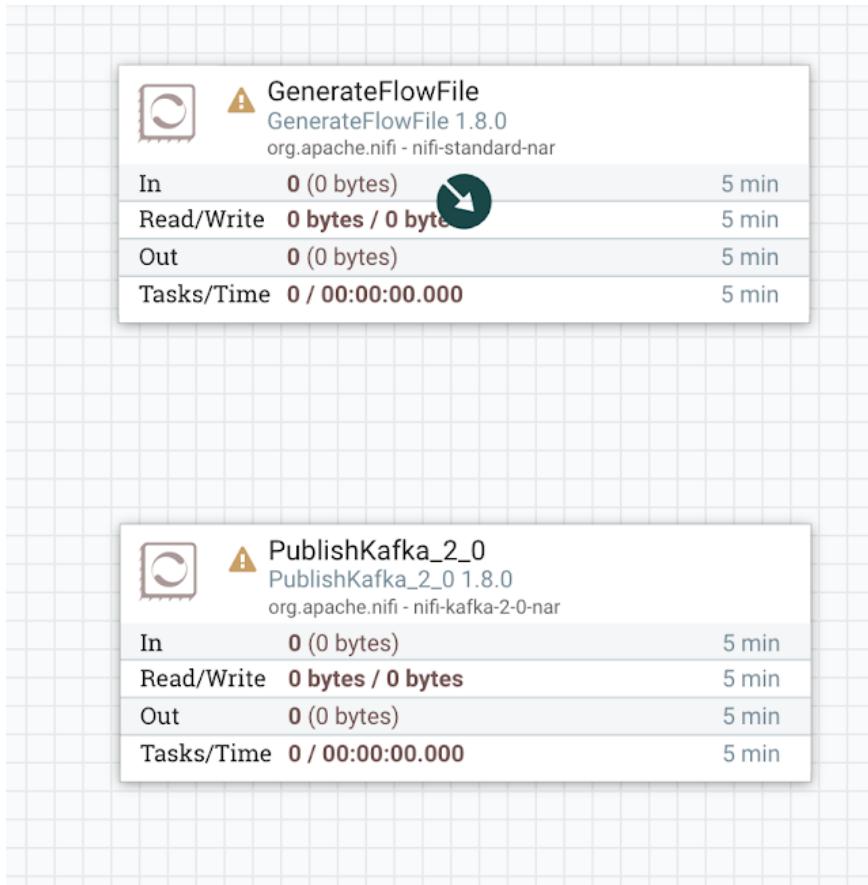
CANCEL ADD

Then click **ADD**.

Now your canvas should look like this:



Now, with the mouse go over the **GenerateFlowFile** step and you should see an arrow icon:



Drag & Drop that icon to the **PublishKafka\_2\_0**, in order to connect the two steps:

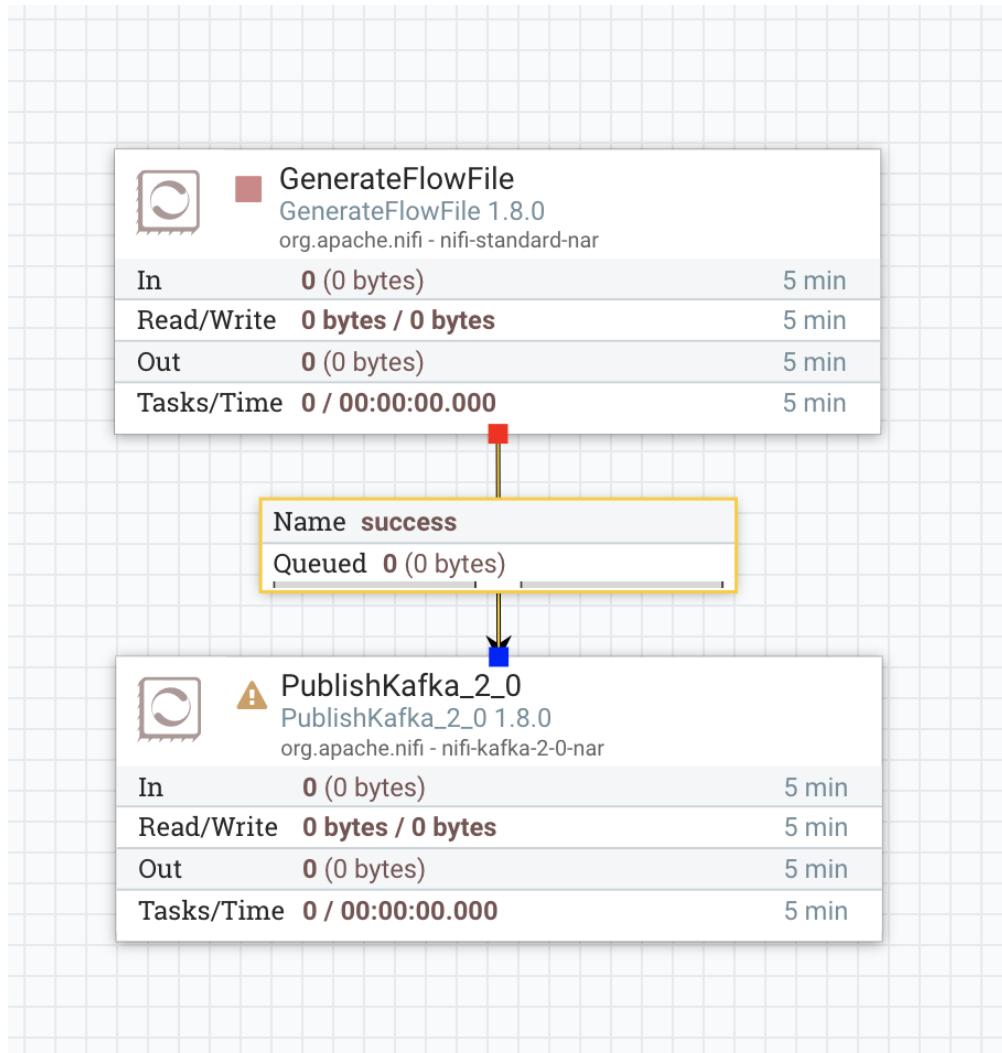
### Create Connection

DETAILS      SETTINGS

From Processor <b>GenerateFlowFile</b> GenerateFlowFile	To Processor <b>PublishKafka_2_0</b> PublishKafka_2_0
Within Group NiFi Flow	Within Group NiFi Flow
For Relationships <input checked="" type="checkbox"/> success	

CANCEL      ADD

Then click **ADD**, and you should see something like this:



Now click on the **PublishKafka\_2\_0** in order to configure it.

## Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Name: PublishKafka\_2\_0     Enabled

Automatically Terminate Relationships ?  
 failure  
Any FlowFile that cannot be sent to Kafka will be routed to this Relationship  
 success  
FlowFiles for which all content was sent to Kafka.

Id: 89654208-96d9-34f2-9267-376199588d18

Type: PublishKafka\_2\_0 1.8.0

Bundle: org.apache.nifi - nifi-kafka-2-0-nar

Penalty Duration ?: 30 sec

Yield Duration ?: 1 sec

Bulletin Level ?: WARN

CANCEL    APPLY

Go to the **PROPERTIES** tab:

### Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS																														
Required field																																	
<table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>Kafka Brokers</td><td>cdedge-95daa600.cdh-cluster.internal:9092</td></tr> <tr><td>Security Protocol</td><td>PLAINTEXT</td></tr> <tr><td>Kerberos Service Name</td><td>No value set</td></tr> <tr><td>Kerberos Credentials Service</td><td>No value set</td></tr> <tr><td>Kerberos Principal</td><td>No value set</td></tr> <tr><td>Kerberos Keytab</td><td>No value set</td></tr> <tr><td>SSL Context Service</td><td>No value set</td></tr> <tr><td>Topic Name</td><td>IOT_Stream</td></tr> <tr><td>Delivery Guarantee</td><td>Best Effort</td></tr> <tr><td>Use Transactions</td><td>false</td></tr> <tr><td>Attributes to Send as Headers (Regex)</td><td>No value set</td></tr> <tr><td>Message Header Encoding</td><td>UTF-8</td></tr> <tr><td>Kafka Key</td><td>No value set</td></tr> <tr><td>Key Attribute Encoding</td><td>UTF-8 Encoded</td></tr> </tbody> </table>		Property	Value	Kafka Brokers	cdedge-95daa600.cdh-cluster.internal:9092	Security Protocol	PLAINTEXT	Kerberos Service Name	No value set	Kerberos Credentials Service	No value set	Kerberos Principal	No value set	Kerberos Keytab	No value set	SSL Context Service	No value set	Topic Name	IOT_Stream	Delivery Guarantee	Best Effort	Use Transactions	false	Attributes to Send as Headers (Regex)	No value set	Message Header Encoding	UTF-8	Kafka Key	No value set	Key Attribute Encoding	UTF-8 Encoded	<span style="border: 1px solid #ccc; padding: 2px;">+</span>	
Property	Value																																
Kafka Brokers	cdedge-95daa600.cdh-cluster.internal:9092																																
Security Protocol	PLAINTEXT																																
Kerberos Service Name	No value set																																
Kerberos Credentials Service	No value set																																
Kerberos Principal	No value set																																
Kerberos Keytab	No value set																																
SSL Context Service	No value set																																
Topic Name	IOT_Stream																																
Delivery Guarantee	Best Effort																																
Use Transactions	false																																
Attributes to Send as Headers (Regex)	No value set																																
Message Header Encoding	UTF-8																																
Kafka Key	No value set																																
Key Attribute Encoding	UTF-8 Encoded																																
<span>CANCEL</span> <span>APPLY</span>																																	

And add the value for the following properties:

- **Kafka Brokers:** <your-private-ip>:9092 (in my case 10.0.0.4:9092)
- **Topic Name:** IOT\_Stream
- **Use Transactions:** false

Once done, click **APPLY**.

In case of something goes wrong when publishing to kafka topic, we want to catch it in order, so go over the **PublishKafka\_2\_0** click on the arrow icon and drag it outside the component and terminate it into the component itself. The Connection box pop-up, check the **failure** box as described below:

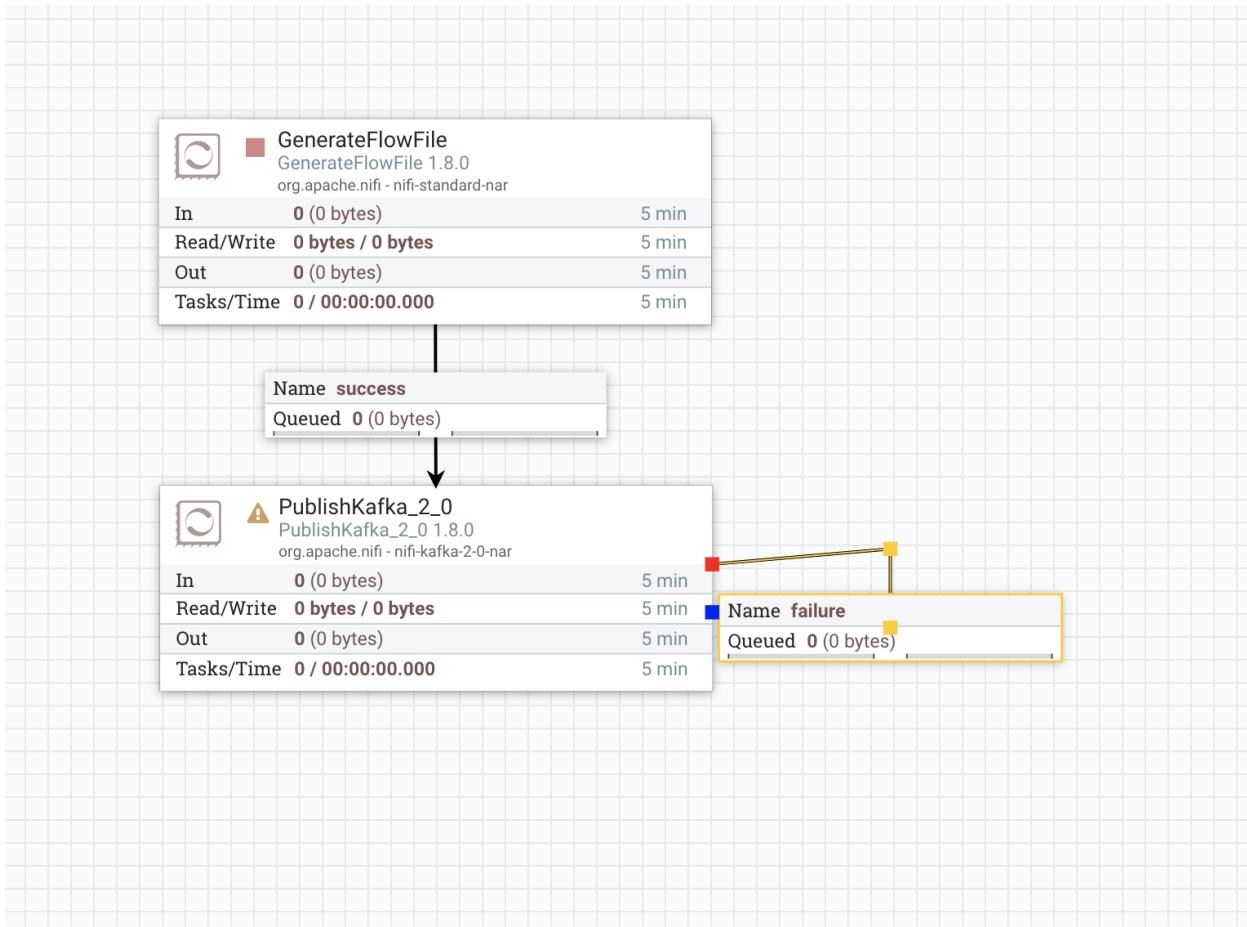
### Create Connection

DETAILS      SETTINGS

From Processor <b>PublishKafka_2_0</b> PublishKafka_2_0	To Processor <b>PublishKafka_2_0</b> PublishKafka_2_0
Within Group NiFi Flow	Within Group NiFi Flow
For Relationships	
<input checked="" type="checkbox"/> failure	
<input type="checkbox"/> success	

CANCEL      ADD

Then click **ADD**, and you should get this:



For sake of simplicity in this lab we have just added a self-loop in case of failure, but for real project is recommended to add an attribute to retry “x” time and after the number “x” tries have been failed then have additional steps for notifying the issue (send email, store in a file, log in APM apps, etc).

We can now terminate this flow by clicking on the PublishKafka\_2\_0 step, go to the **SETTINGS** tab and check the **success** box:

## Configure Processor

SETTINGS   SCHEDULING   PROPERTIES   COMMENTS

Name: PublishKafka\_2\_0    Enabled

Id: 3c367ca7-0168-1000-e166-4229f9fc6bec

Type: PublishKafka\_2\_0 1.8.0

Bundle: org.apache.nifi - nifi-kafka-2-0-nar

Penalty Duration: 30 sec   Yield Duration: 1 sec

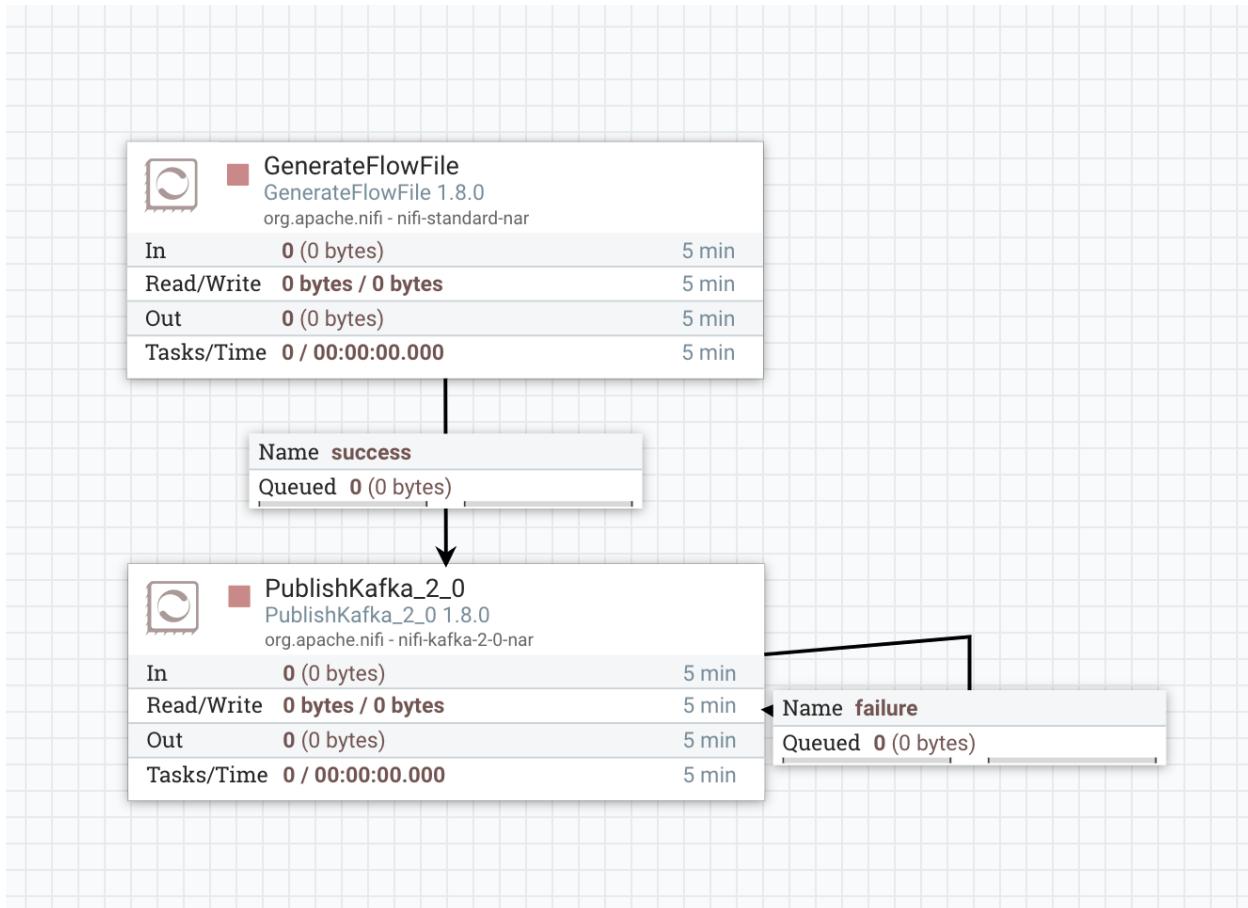
Bulletin Level:

Automatically Terminate Relationships:  failure  
Any FlowFile that cannot be sent to Kafka will be routed to this Relationship.

success  
FlowFiles for which all content was sent to Kafka.

CANCEL   APPLY

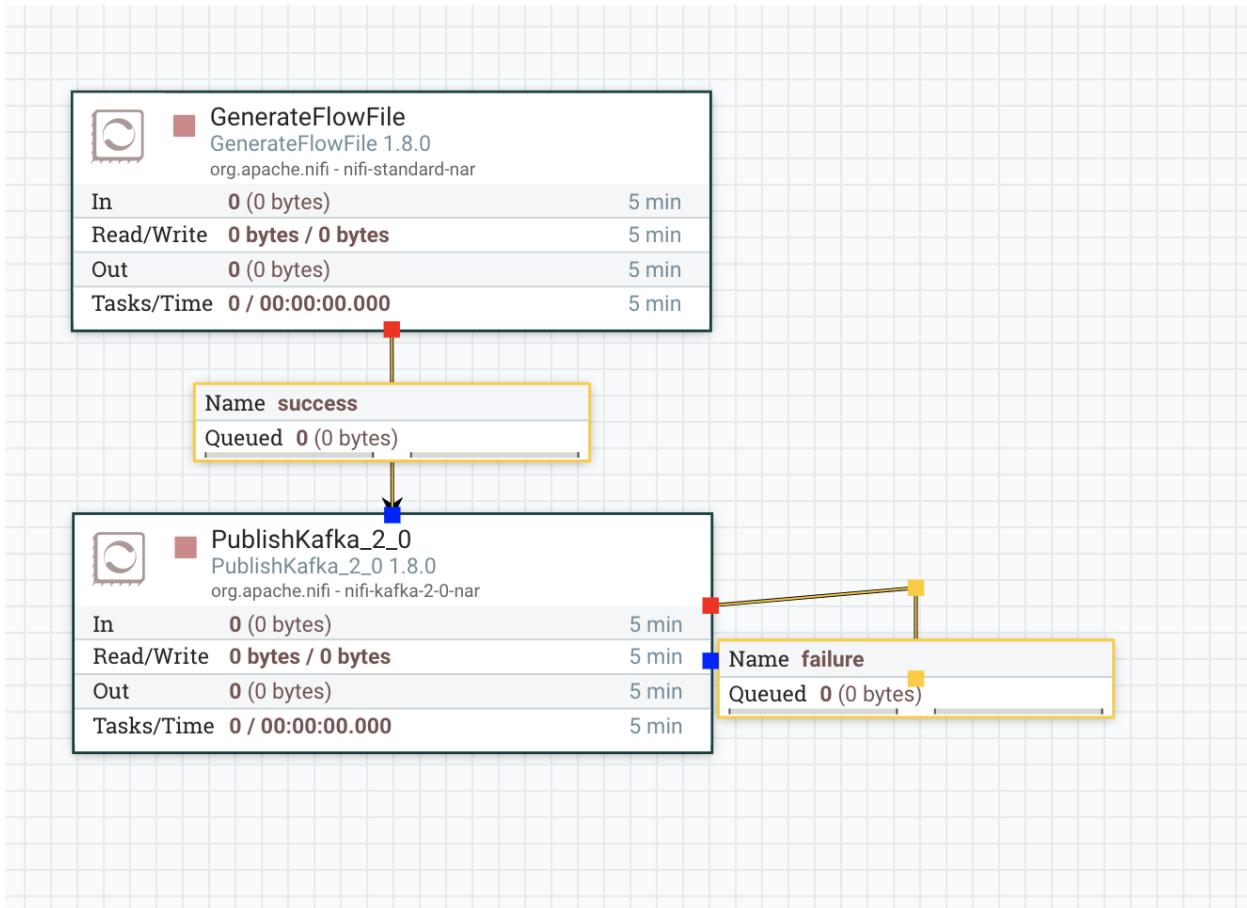
Then the warning icon on the top left corner of this component should be replaced by the “STOP” icon:



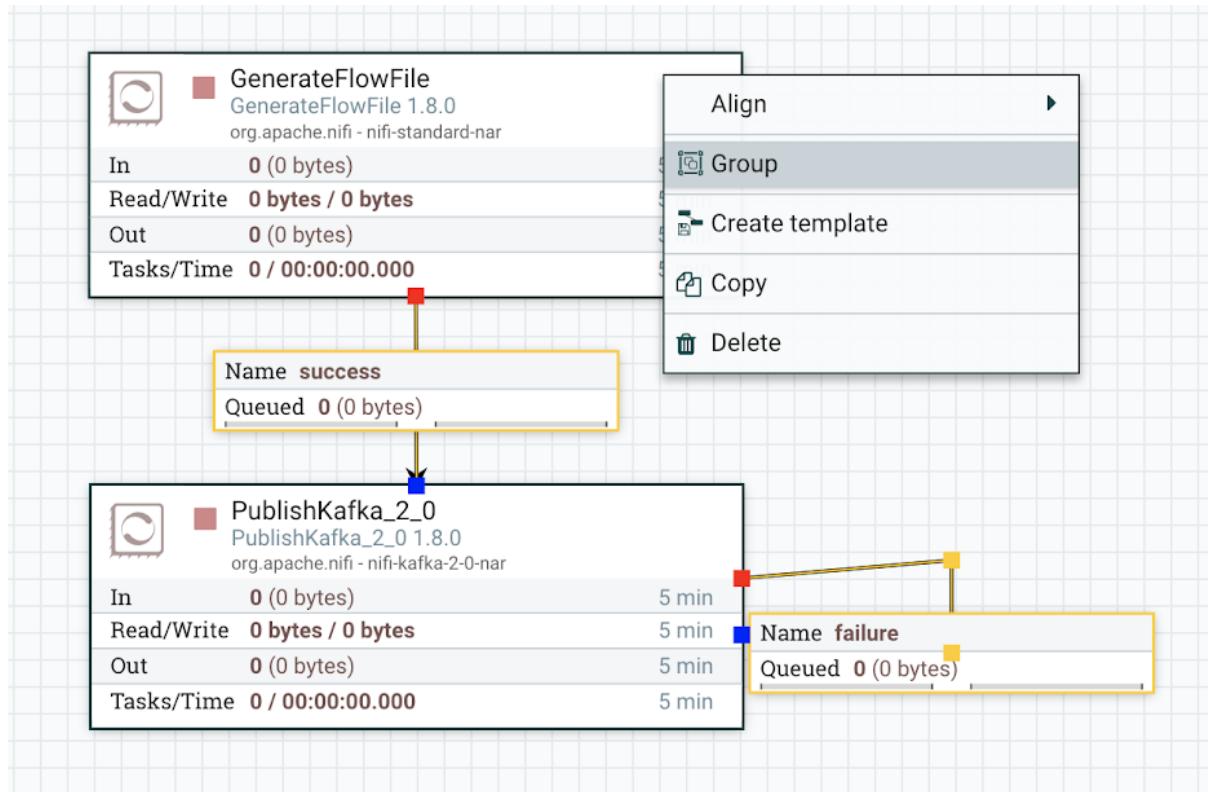
Now the flow for generating random sensors data reading is ready to go, but before run it let's create also the flow for ingesting the data into Kudu, as described in the next session.

But before doing that, let's organize better what we have done so far.

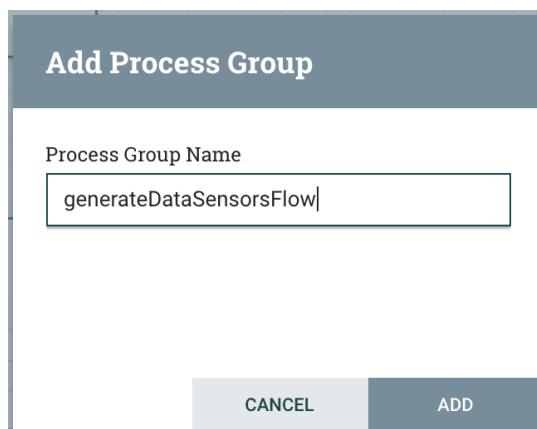
With the mouse select a point in the canvas, press SHIFT and hold down the mouse and select your flow:



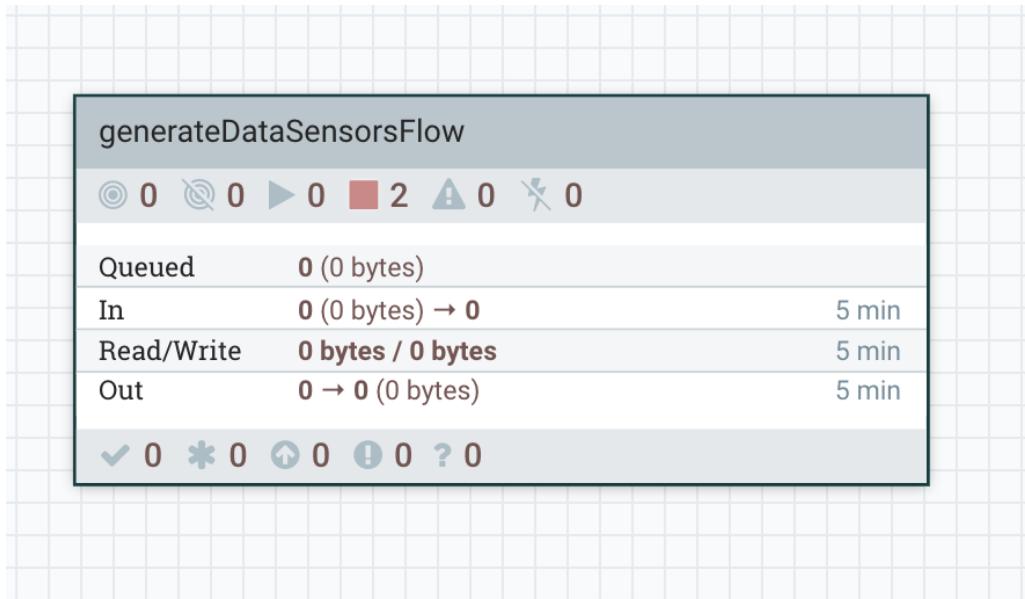
Then right click and click **Group**



Give it a name, in my case *generateDataSensorsFlow*:



And click **ADD**, we should get the following:



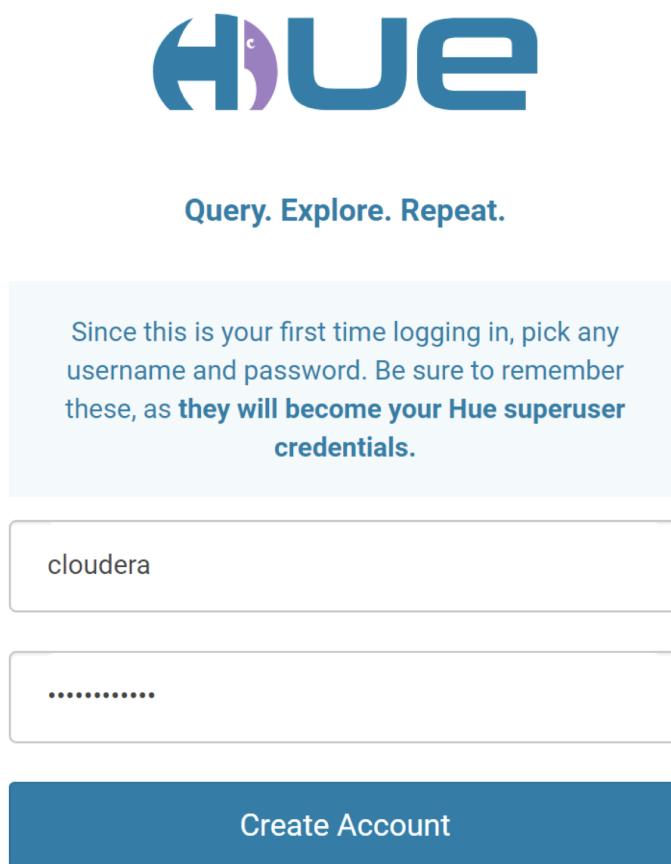
Now we are ready to move forward.

## Lab 4: Define the Kudu tables for the Connected Thing time series data

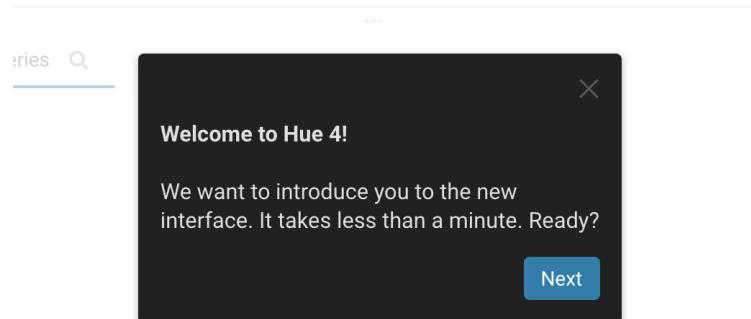
Use the following procedure to access the HUE interface in order to create the table that will be used to store the data from the remote Things.

Connect to Hue using the following url (don't forget to change the ip address):  
<http://40.76.204.159:8888>

If it's the first time to access HUE, then you will need to create an account (I'd suggest you use **cloudera** and **Cloudera\_123**)



Close with pop-up window:



Within the Impala Query Editor type the following SQL statement:

```
CREATE TABLE sensor_table
(
    sensorID INT,
    sensorValue INT,
    sensorTimestamp TIMESTAMP,
    PRIMARY KEY (sensorID)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

The above statement is going to create a table, called `sensor_table`, with 3 fields, the `sensorID` as primary key, and 16 hash partitions stored as Kudu.

Click the run icon



The screenshot shows the Hue interface for running Impala queries. In the top navigation bar, 'HUE' is selected. The main area is titled 'Impala' and shows a query editor with the following code:

```
1 CREATE TABLE sensor_table
2 (
3     sensorID INT,
4     sensorValue INT,
5     sensorTimestamp TIMESTAMP,
6     PRIMARY KEY (sensorID)
7 )
8 PARTITION BY HASH PARTITIONS 16
9 STORED AS KUDU;
```

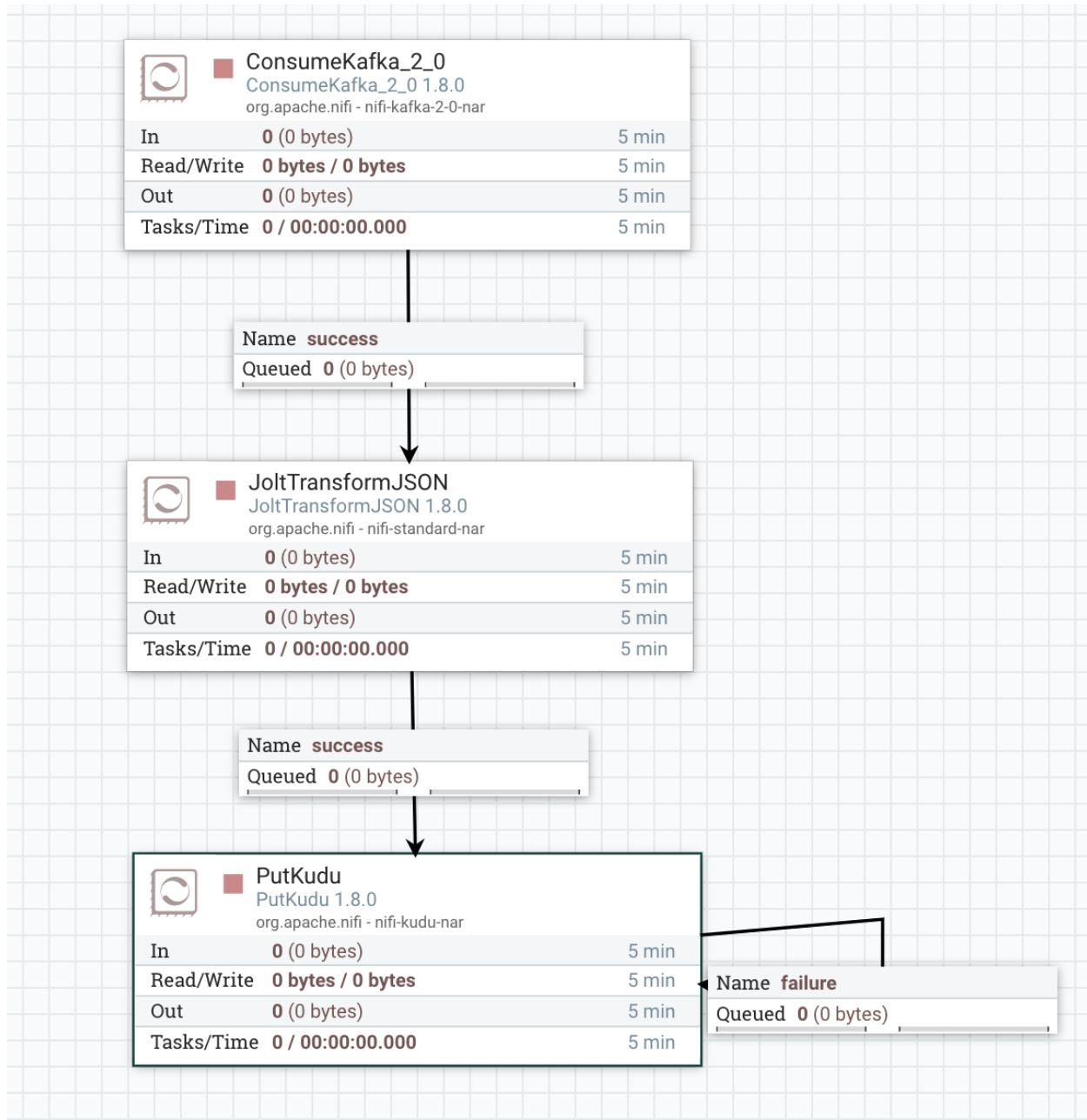
Below the code, a message says 'Success.' and shows a timestamp of '3.37s'. At the bottom, the 'Query History' tab is active, displaying the query just run: 'CREATE TABLE sensor\_table ( sensorID INT, sensorValue INT, sensorTimestamp TIMESTAMP, PRIMARY KEY (sensorID) ) PARTITION BY HASH PARTITIONS 16 STORED AS KUDU'. There is also a 'Saved Queries' tab.

## Lab 5: Create the real-time ingest pipeline

Now that we have a way to simulate the time series readings from 10,000 sensors and the kudu table to store the data, we can proceed further and create the real-time ingestion pipeline that will take the readings from the remote Things and update the Kudu table.

ingestDataSensors flow

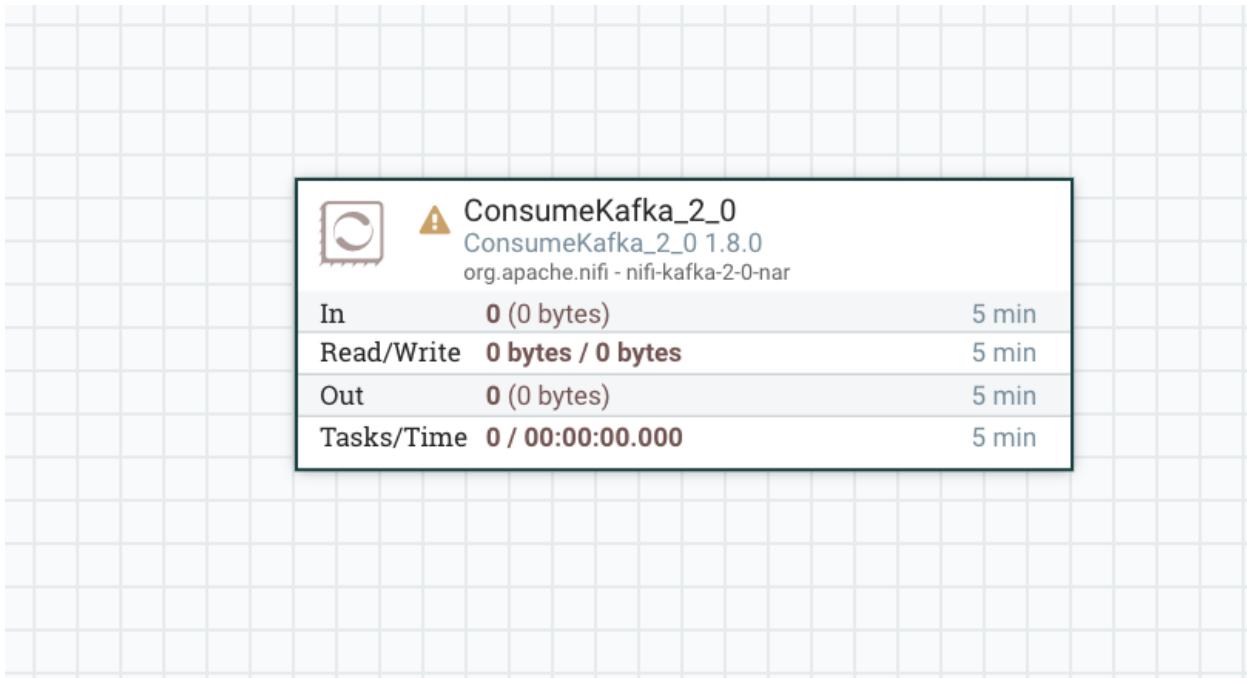
This flow is going to look like the following picture



Now you should be already familiar on how to add a new processor to the canvas, then add the **ConsumeKafka\_2\_0** processor:

The screenshot shows the 'Add Processor' dialog in NiFi. The search bar at the top contains the text 'consumeKafka\_2\_0'. Below the search bar, there is a table with the following columns: Source, Type, Version, and Tags. The 'Type' column is currently sorted by Type (A-Z). The 'Tags' column is also sorted by Tag (A-Z). The 'Tags' column for the 'ConsumeKafka\_2\_0' processor includes 'PubSub, Consume, Ingest, 2.0, ...'. The 'Version' column shows '1.8.0'. The 'Source' dropdown is set to 'all groups'. The 'Tags' dropdown is also set to 'all groups'. On the left side of the dialog, there is a sidebar with various processor categories listed as buttons: amazon, attributes, avro, aws, consume, csv, delete, fetch, get, hadoop, ingest, ingress, insert, json, kafka, listen, logs, message, pubsub, put, record, restricted, source, text, update. At the bottom of the dialog, there is a description of the 'ConsumeKafka\_2\_0' processor: 'Consumes messages from Apache Kafka specifically built against the Kafka 2.0 Consumer API. The complementary NiFi processor for sending messages is PublishKafka\_2\_0.' There are two buttons at the bottom right: 'CANCEL' and 'ADD'.

Click **ADD**



Double click on the **ConsumeKafka\_2\_0** and go to the **PROPERTIES** tab

### Configure Processor

**Required field**

Property	Value
Kafka Brokers	cdedge-95daa600.cdh-cluster.internal:9092
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Topic Name(s)	IOT_Stream
Topic Name Format	names
Honor Transactions	true
Group ID	nifi_consumer
Offset Reset	latest
Key Attribute Encoding	UTF-8 Encoded
Message Demarcator	No value set

**CANCEL** **APPLY**

And add the value for the following properties:

- **Kafka Brokers:** <your-private-ip>:9092 (in my example: 10.0.0.4:9092)
- **Topic Name(s):** IOT\_Stream
- **Group ID:** nifi\_consumer

Once done, click **APPLY**

Now we need to transform the message received by Kafka consumer in order to adapt it to the kudu schema table. In particular we need to transform all the fields in lower letter.

Add a **JoltTransformJSON** to the canvas:

### Add Processor

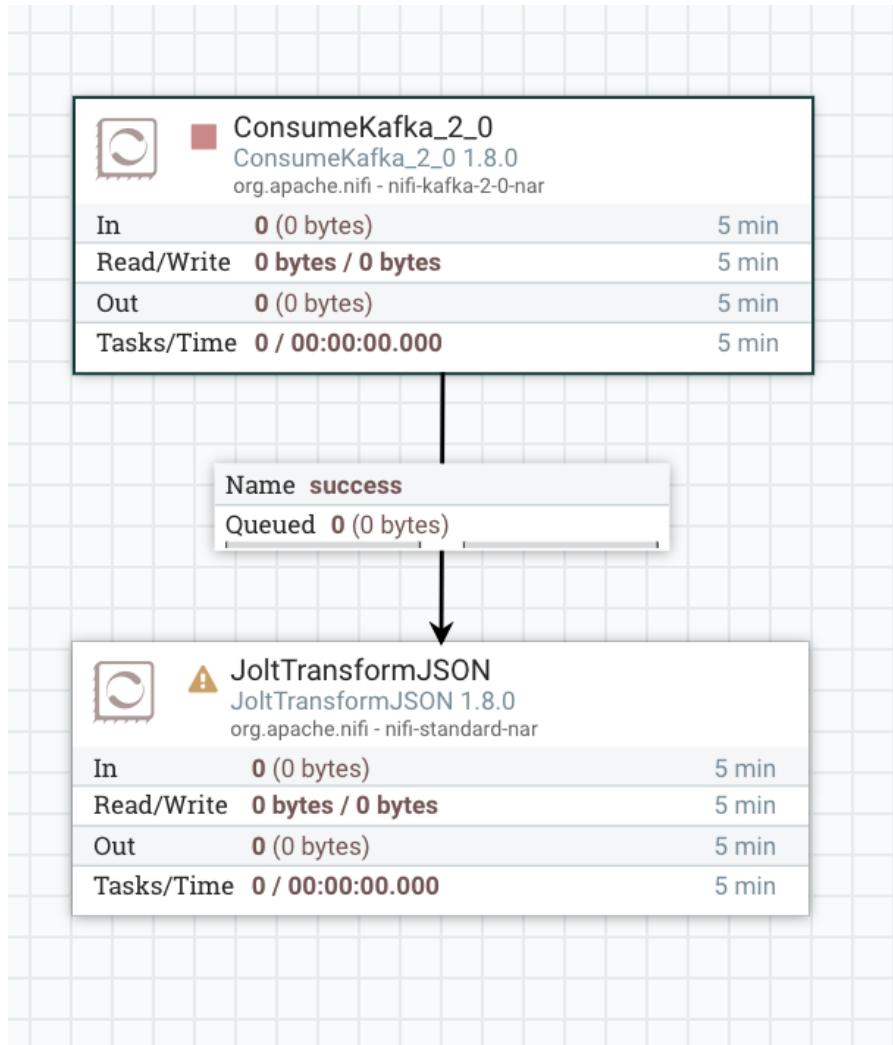
Source	Type	Version	Tags
all groups	JoltTransformJSON	1.8.0	transform, chainr, defaultr, shiftr...
	JoltTransformRecord	1.8.0	transform, chainr, record, defaul...

**JoltTransformJSON 1.8.0** org.apache.nifi - nifi-standard-nar

Applies a list of Jolt specifications to the flowfile JSON payload. A new FlowFile is created with transformed content and is routed to the 'success' relationship. If the JSON transform fails, the original FlowFile is routed to the 'failure' relationship.

**CANCEL** **ADD**

Click **ADD** and connect the kafka consumer to this component:



Connect the kafka consumer to the transform step, now you should know how to do it.  
Double click on the **JoltTransformJSON** component and go to the **SETTINGS** tab

## Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Name: JoltTransformJSON     Enabled

Id: 3d3b14cb-0168-1000-65ab-94ffa3c12931

Type: JoltTransformJSON 1.8.0

Bundle: org.apache.nifi - nifi-standard-nar

Penalty Duration: 30 sec    Yield Duration: 1 sec

Bulletin Level:

Automatically Terminate Relationships:  failure  
If a FlowFile fails processing for any reason (for example, the FlowFile is not valid JSON), it will be routed to this relationship  
 success  
The FlowFile with transformed content will be routed to this relationship

CANCEL    APPLY

Check the **failure** box and go to the **PROPERTIES** tab

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

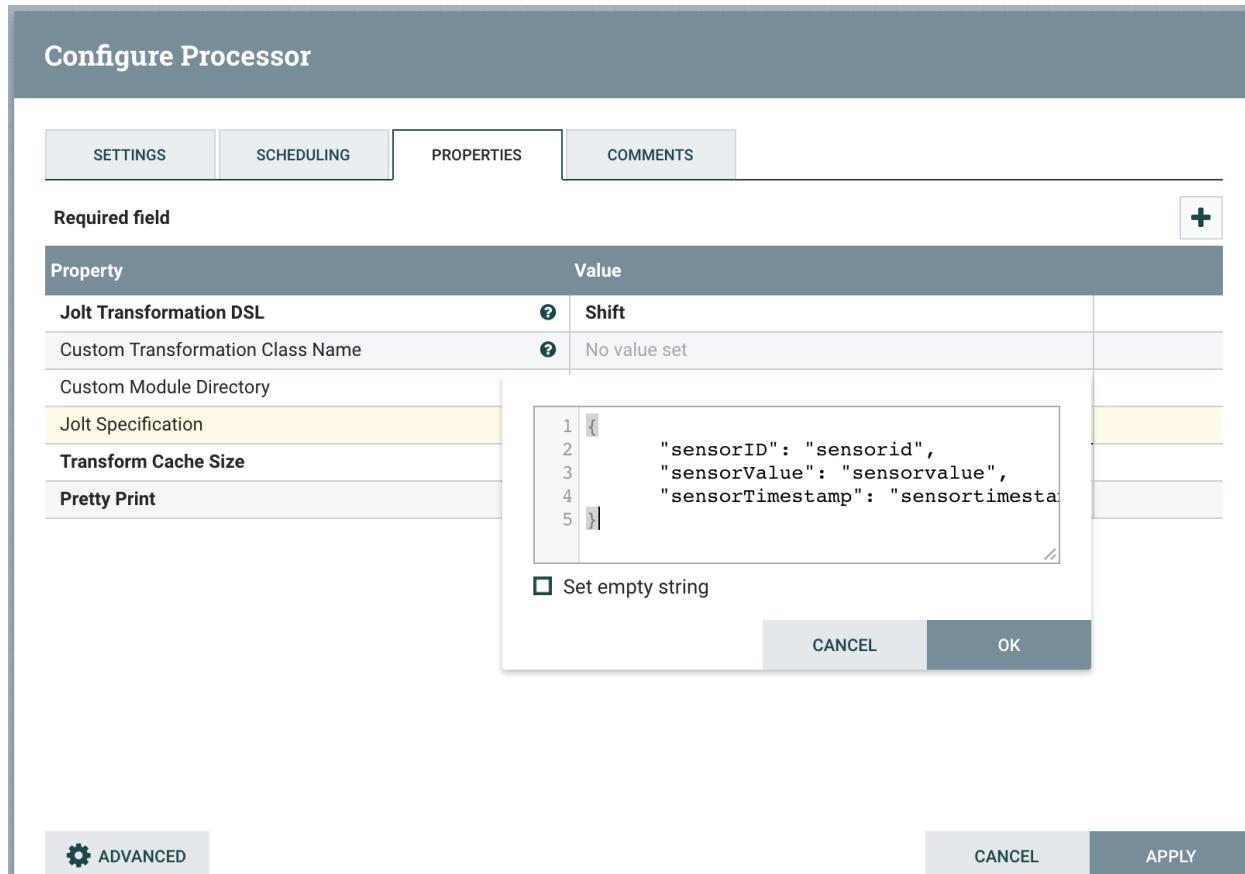
Required field +

Property	Value
Jolt Transformation DSL	Shift
Custom Transformation Class Name	No value set
Custom Module Directory	No value set
Jolt Specification	{ "sensorID": "sensorid", "sensorValue": "sensorvalue", "senso...
Transform Cache Size	1
Pretty Print	false

ADVANCED CANCEL APPLY

Add the following values:

- **Jolt Transformation DSL:** Shift
- **Jolt Specification:** {  
    "sensorID": "sensorid",  
    "sensorValue": "sensorvalue",  
    "sensorTimestamp": "sensortimestamp"  
}



Then click **OK** and **APPLY**.

Now add a **PutKudu** processor to the canvas

Add Processor

Source	Type	Version	Tags
all groups	PutKudu	1.8.0	database, NoSQL, record, HDFS...

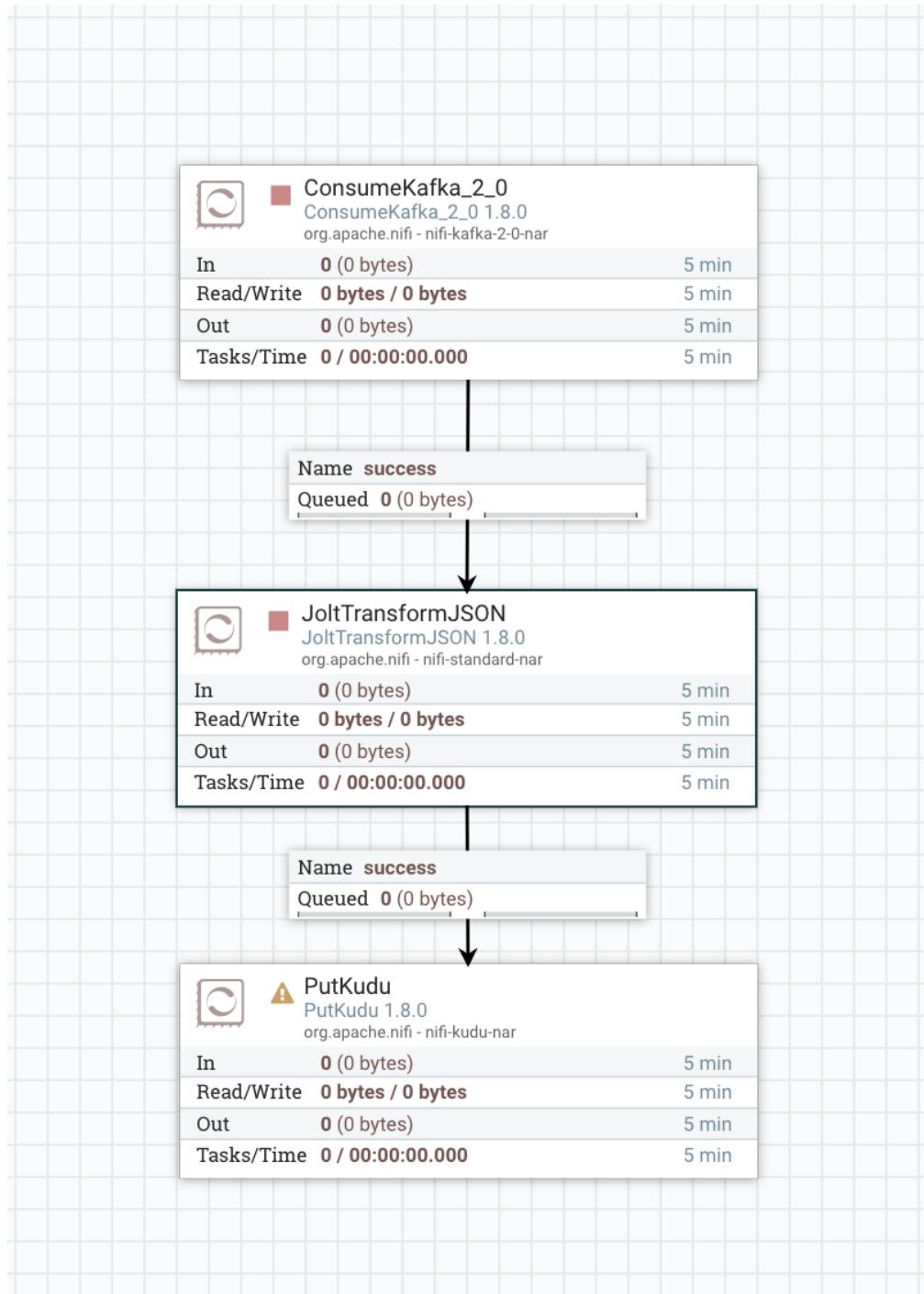
amazon attributes  
avro aws consume  
csv delete fetch  
get hadoop  
ingest ingress  
insert json kafka  
listen logs  
message pubsub  
put record  
restricted source  
text update

**PutKudu 1.8.0 org.apache.nifi - nifi-kudu-nar**

Reads records from an incoming FlowFile using the provided Record Reader, and writes those records to the specified Kudu's table. The schema for the table must be provided in the processor properties or from your source. If any error occurs while reading records from the input, or writing records to Kudu, the FlowFile will be routed to failure

CANCEL ADD

Click **ADD** and connect to the **JoltTransformJSON**



Double click on PutKudu step and go to the **SETTINGS** tab

## Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Name: PutKudu    Enabled:

Id: 79ba58dc-0548-3d54-c320-04e5b3883b5b

Type: PutKudu 1.8.0

Bundle: org.apache.nifi - nifi-kudu-nar

Penalty Duration: 30 sec    Yield Duration: 1 sec

Bulletin Level:

Automatically Terminate Relationships:  failure  
A FlowFile is routed to this relationship if it cannot be sent to Kudu

success  
A FlowFile is routed to this relationship after it has been successfully stored in Kudu

CANCEL    APPLY

Check the **success** box and go to the **PROPERTIES** tab

### Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS																														
Required field																																	
<table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Kudu Masters</td> <td>?</td> <td>No value set</td> <td></td> </tr> <tr> <td>Table Name</td> <td>?</td> <td>No value set</td> <td></td> </tr> <tr> <td>Skip head line</td> <td>?</td> <td>false</td> <td></td> </tr> <tr> <td>Record Reader</td> <td>?</td> <td>No value set</td> <td></td> </tr> <tr> <td>Insert Operation</td> <td>?</td> <td>INSERT</td> <td></td> </tr> <tr> <td>Flush Mode</td> <td>?</td> <td>AUTO_FLUSH_BACKGROUND</td> <td></td> </tr> <tr> <td>Batch Size</td> <td>?</td> <td>100</td> <td></td> </tr> </tbody> </table>				Property	Value	Kudu Masters	?	No value set		Table Name	?	No value set		Skip head line	?	false		Record Reader	?	No value set		Insert Operation	?	INSERT		Flush Mode	?	AUTO_FLUSH_BACKGROUND		Batch Size	?	100	
Property	Value																																
Kudu Masters	?	No value set																															
Table Name	?	No value set																															
Skip head line	?	false																															
Record Reader	?	No value set																															
Insert Operation	?	INSERT																															
Flush Mode	?	AUTO_FLUSH_BACKGROUND																															
Batch Size	?	100																															
		CANCEL	APPLY																														

Set:

- **Kudu Masters:** <your-private-ip>:7051 (in my case 10.0.0.4:7051)
- **Table Name:** impala::default.sensor\_table
- **Insert Operation:** UPSERT

Then for the **Record Reader** ...

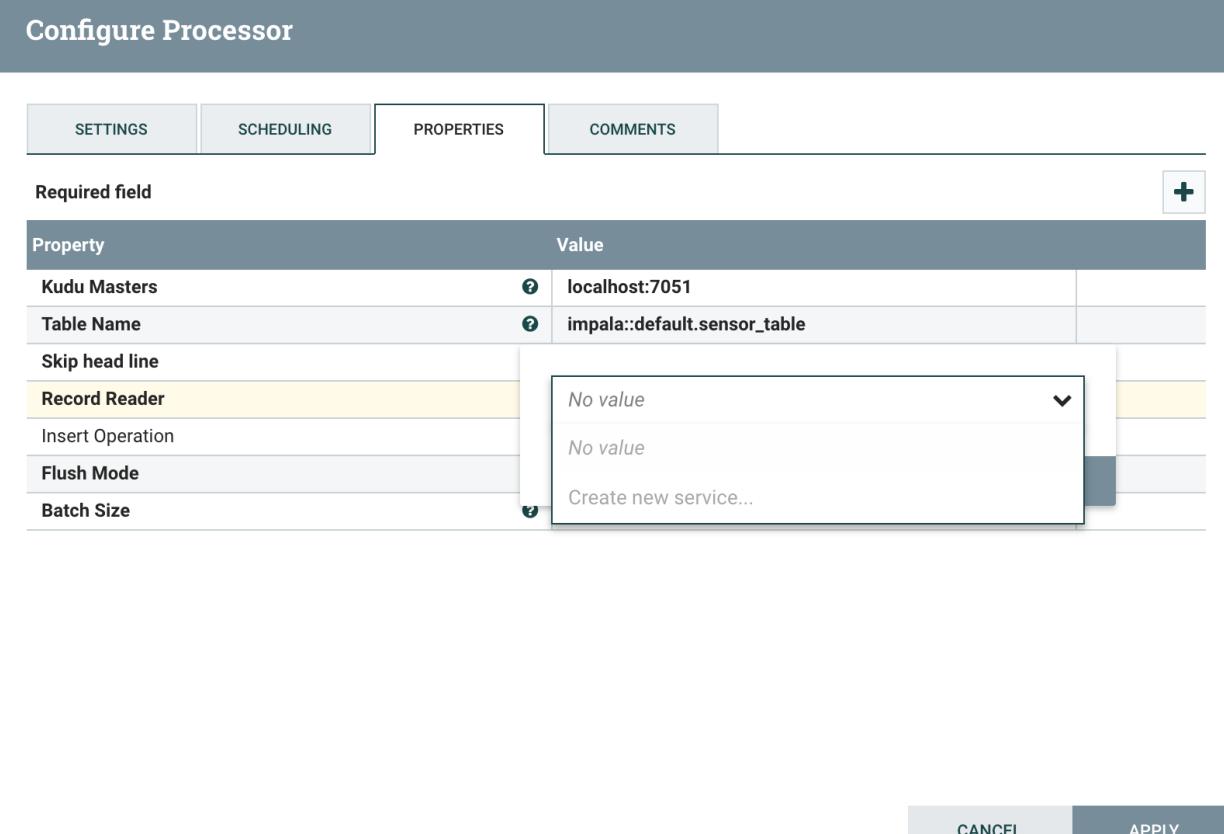
Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

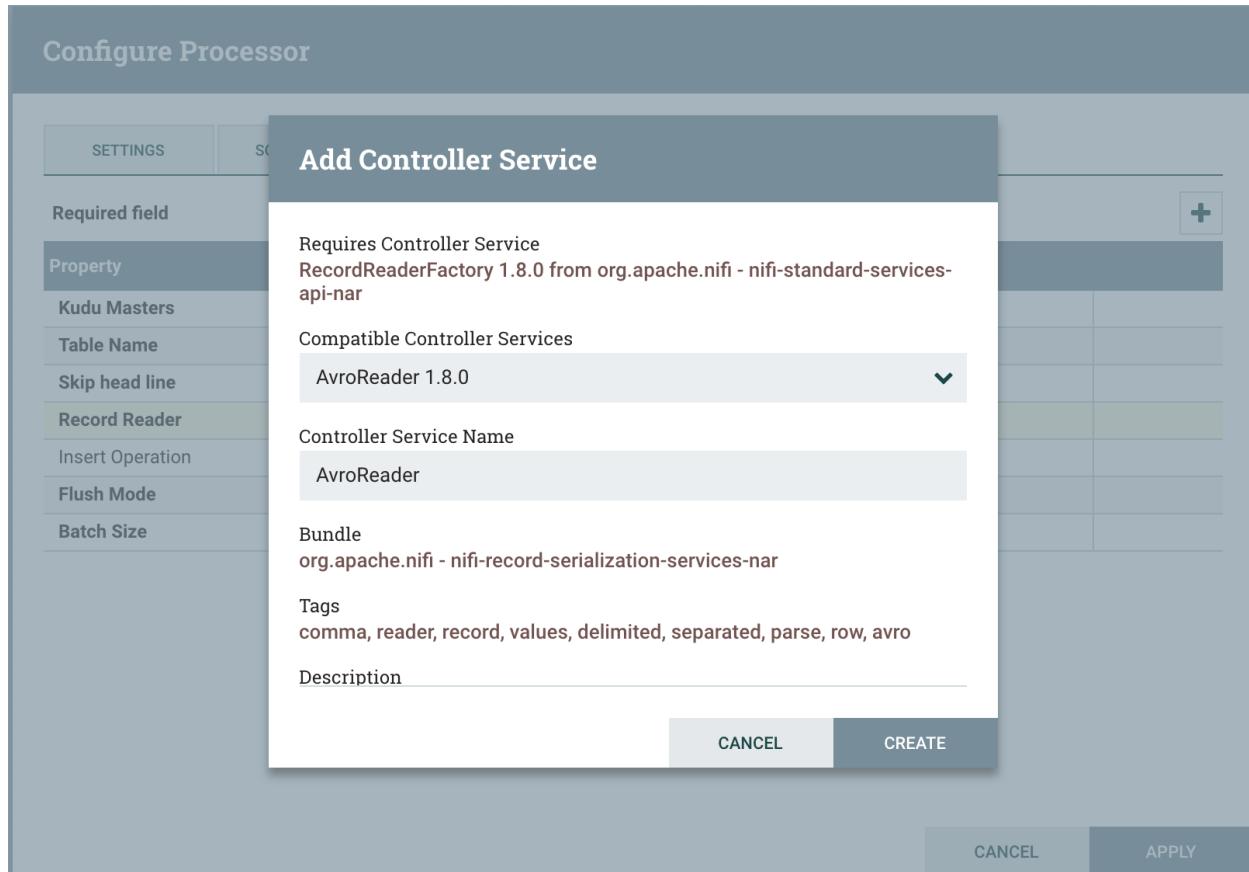
Required field

Property	Value
Kudu Masters	localhost:7051
Table Name	impala::default.sensor_table
Skip head line	
Record Reader	No value
Insert Operation	No value
Flush Mode	
Batch Size	Create new service...

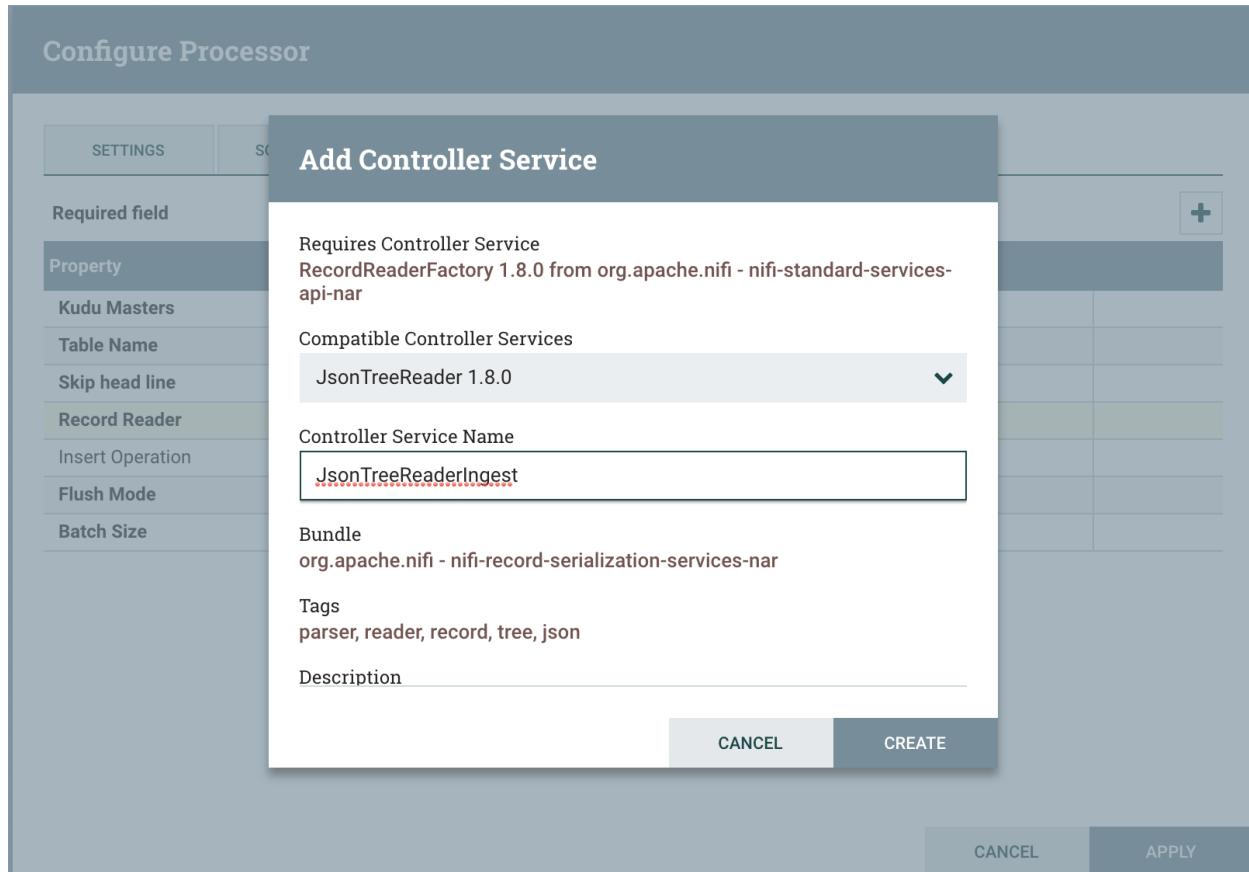
CANCEL APPLY



click on **Create new services...**



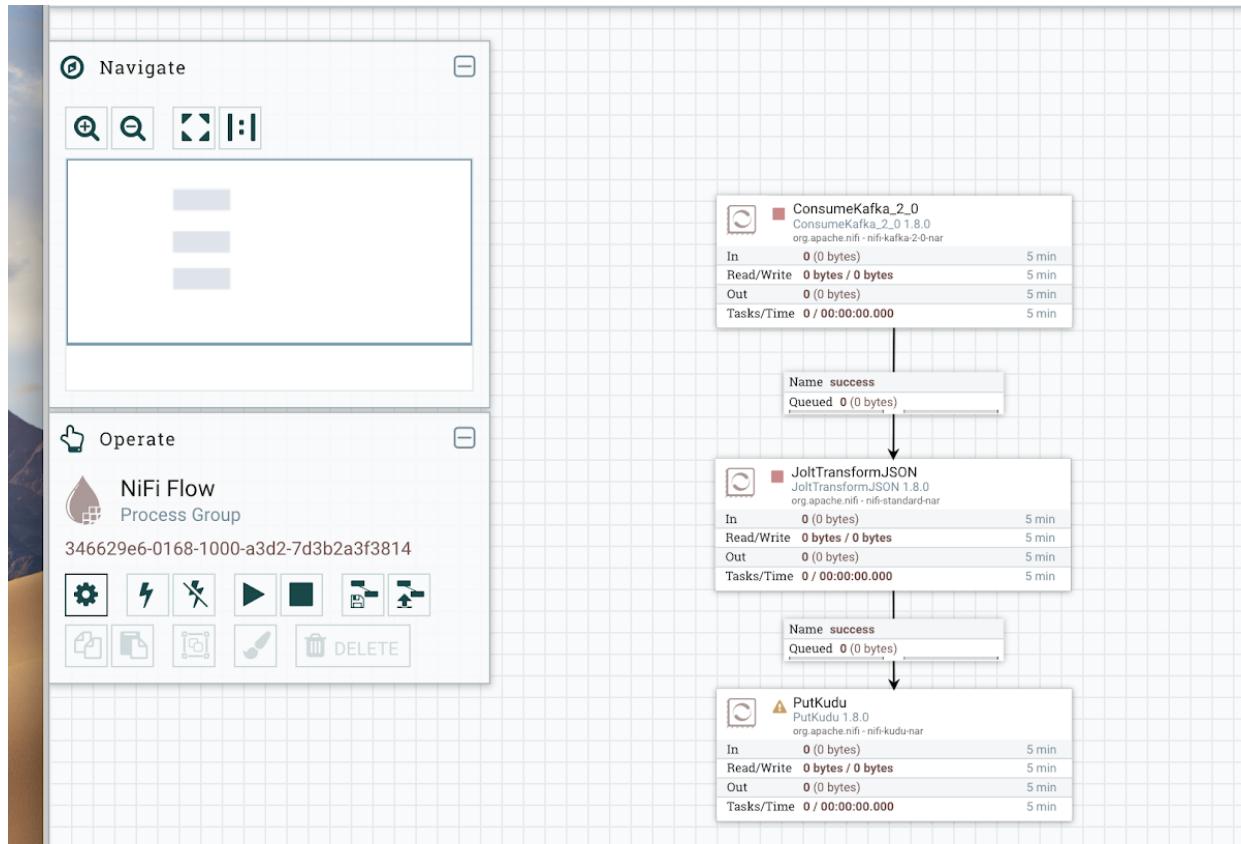
In the **Compatible Controller Services** select **JsonTreeReader** and give it a name  
(JsonTreeReaderIngest)



Then click **CREATE**.

Then click **APPLY**.

Deselect the **PutKudu** processor by clicking on the canvas, and then click the **Configuration** icon in the **Operate** box



Then go to the **CONTROLLER SERVICES** tab

Name	Type	Bundle	State	Scope
JsonTreeReaderIngest	JsonTreeReader 1.8.0	org.apache.nifi - nifi-record-serialization-service...	Invalid	NiFi Flow

Click on the **configure** icon on the right end of the **JsonTreeReaderIngest** row and go to the **PROPERTIES** tab

### Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	No value set
Schema Name	\$(schema.name)
Schema Version	No value set
Schema Branch	No value set
Schema Text	\$(avro.schema)
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set

**CANCEL** **APPLY**

In the **Schema Registry** field click to the value field, click **Create new service...**

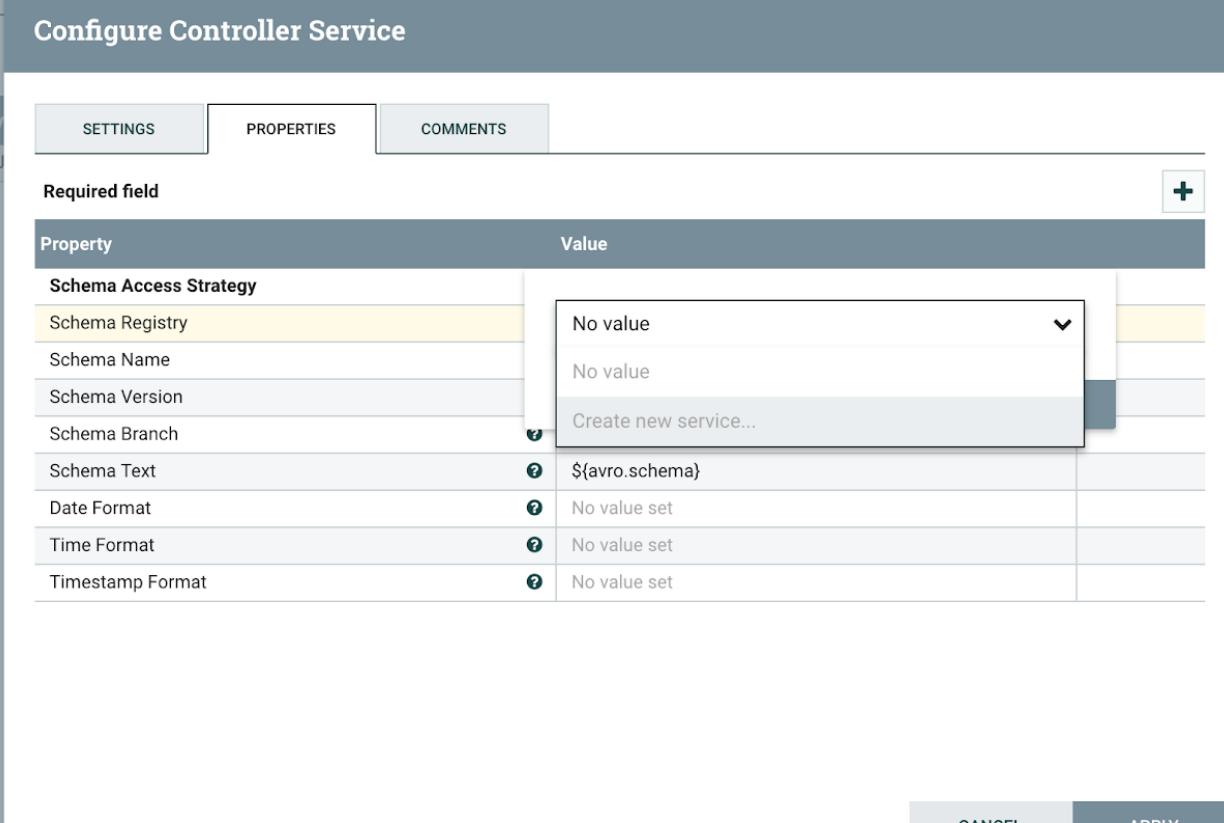
Configure Controller Service

SETTINGS PROPERTIES COMMENTS

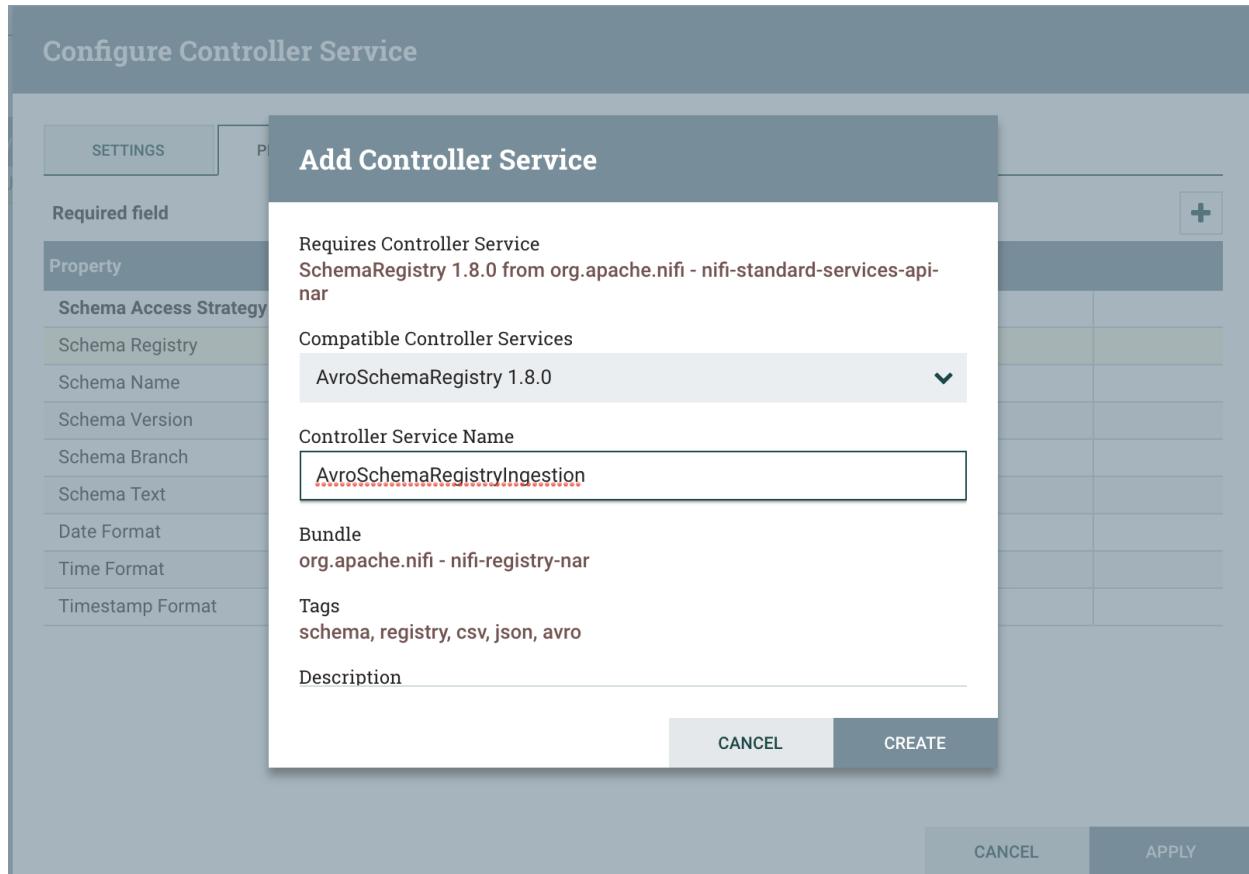
Required field

Property	Value
Schema Access Strategy	No value
Schema Registry	No value
Schema Name	Create new service...
Schema Version	No value set
Schema Branch	No value set
Schema Text	#{avro.schema}
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set

CANCEL APPLY



Select **AvroSchemaRegistry 1.8.0** and give it a name like *AvroSchemaRegistryIngestion*.



Then click **CREATE**.

In the **Schema Name** field set *demo-schema*

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field

Property Value

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	
Schema Name	demo-schema
Schema Version	
Schema Branch	
Schema Text	
Date Format	
Time Format	
Timestamp Format	

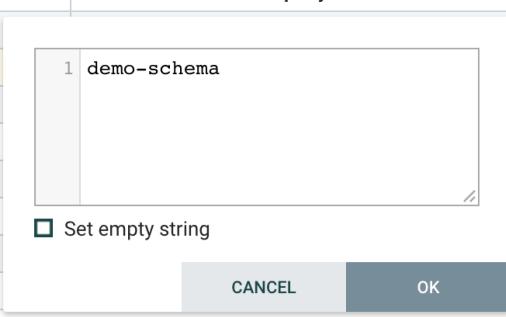
+ →

1 demo-schema

Set empty string

CANCEL OK

CANCEL APPLY



Click **OK**

Now click on the arrow icon in the **Schema Registry** row

Configure Controller Service

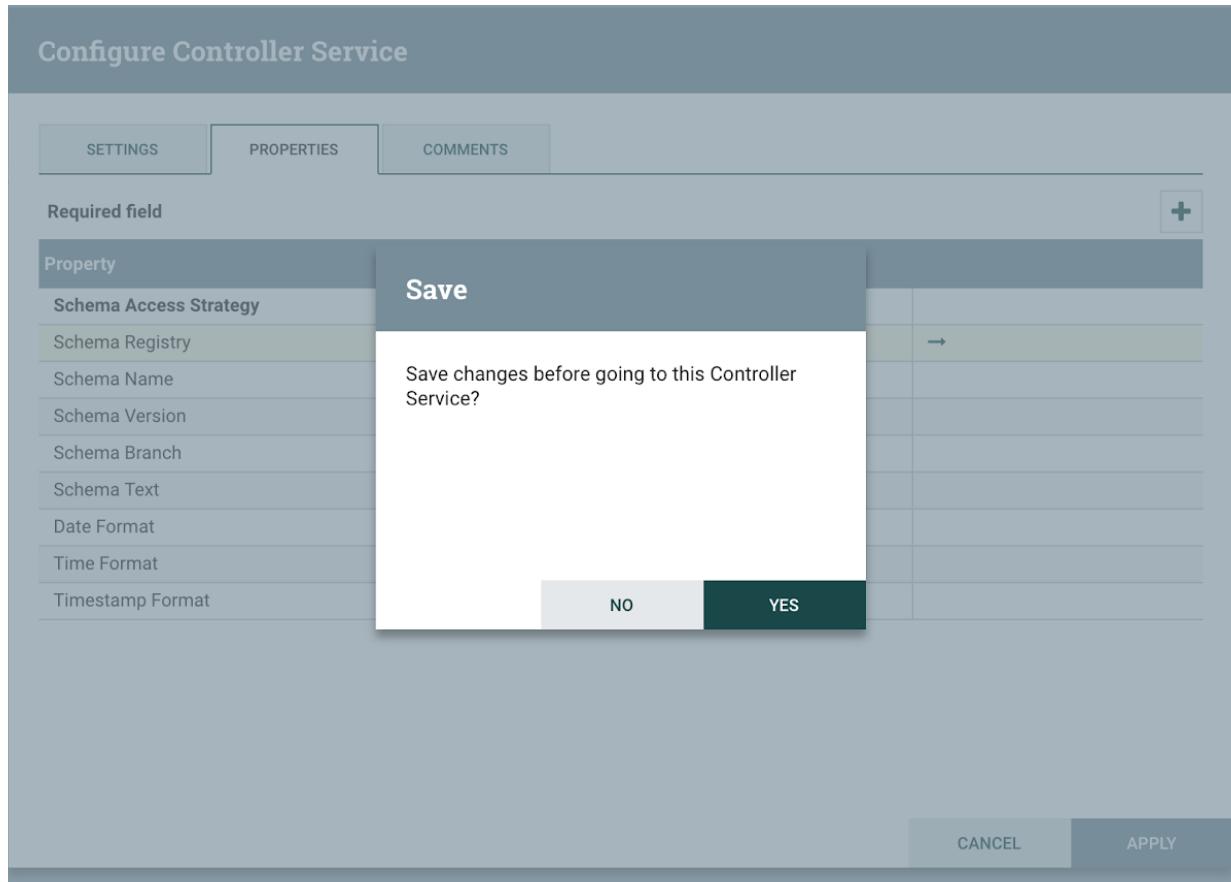
SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	AvroSchemaRegistryIngestion
Schema Name	demo-schema
Schema Version	No value set
Schema Branch	No value set
Schema Text	\${avro.schema}
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set

CANCEL APPLY

Save the change



Click YES

The screenshot shows the 'NiFi Flow Configuration' interface. At the top, there are two tabs: 'GENERAL' and 'CONTROLLER SERVICES', with 'CONTROLLER SERVICES' being active. Below the tabs is a table listing controller services:

Name	Type	Bundle	State	Scope
AvroSchemaRegistryIngestion	AvroSchemaRegistry 1.8.0	org.apache.nifi - nifi-registry-nar	Disabled	NIFI Flow
JsonTreeReaderIngest	JsonTreeReader 1.8.0	org.apache.nifi - nifi-record-serialization-service..	Invalid	NIFI Flow

Click on the **Configure** icon for the **AvroSchemaRegistryIngestion**, then click on the plus sign

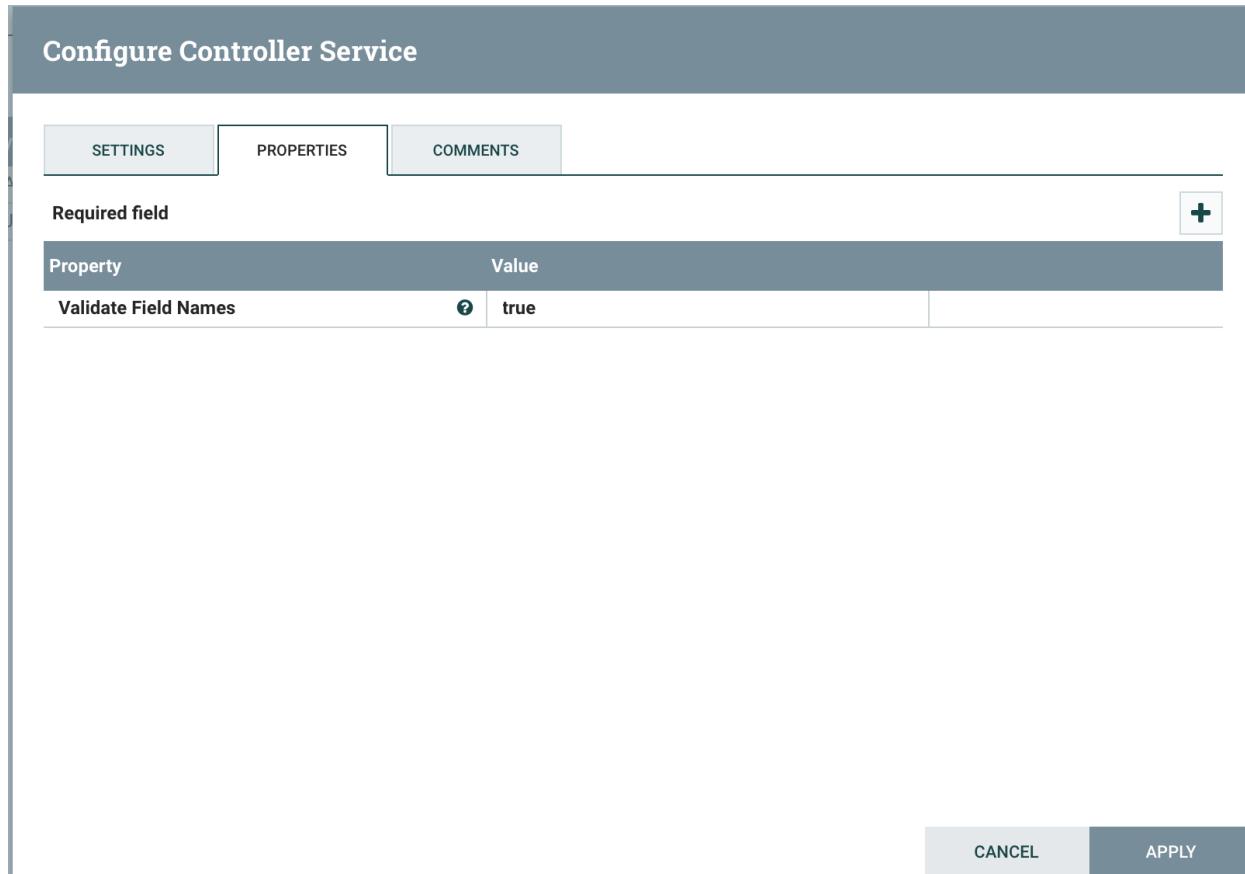
### Configure Controller Service

SETTINGS PROPERTIES COMMENTS

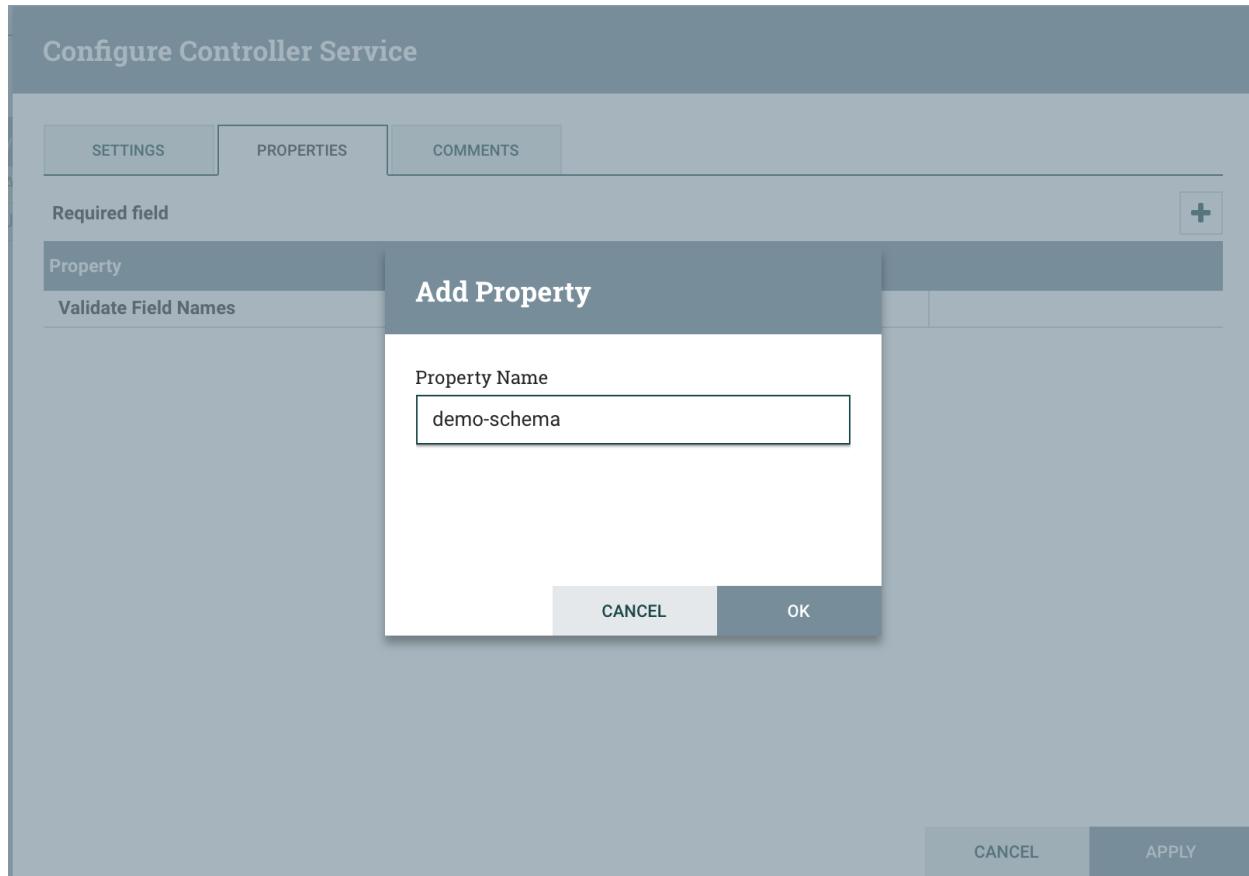
Required field +

Property	Value
Validate Field Names	<span style="border: 1px solid #ccc; padding: 2px;">?</span> true

CANCEL APPLY



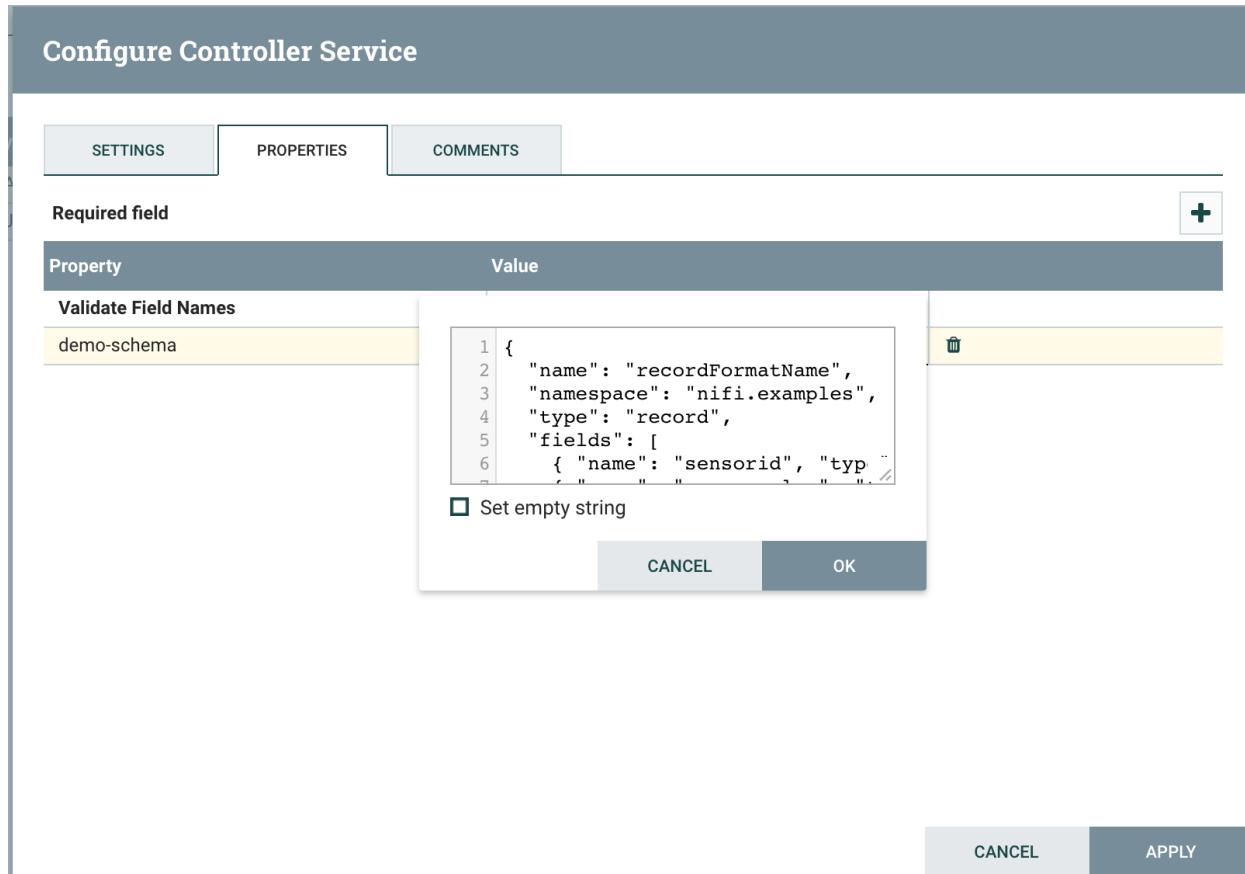
Add *demo-schema*



Then click **OK**.

As the Value add the following:

```
{  
  "name": "recordFormatName",  
  "namespace": "nifi.examples",  
  "type": "record",  
  "fields": [  
    { "name": "sensorid", "type": "double" },  
    { "name": "sensorvalue", "type": "double" },  
    { "name": "sensortimestamp", "type": "double" }  
  ]  
}
```



And click **OK**

### Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value	
Validate Field Names	?	true
demo-schema	?	{ "name": "recordFormatName", "namespace": "nifi...." <span style="color: red;">✖</span>

CANCEL APPLY

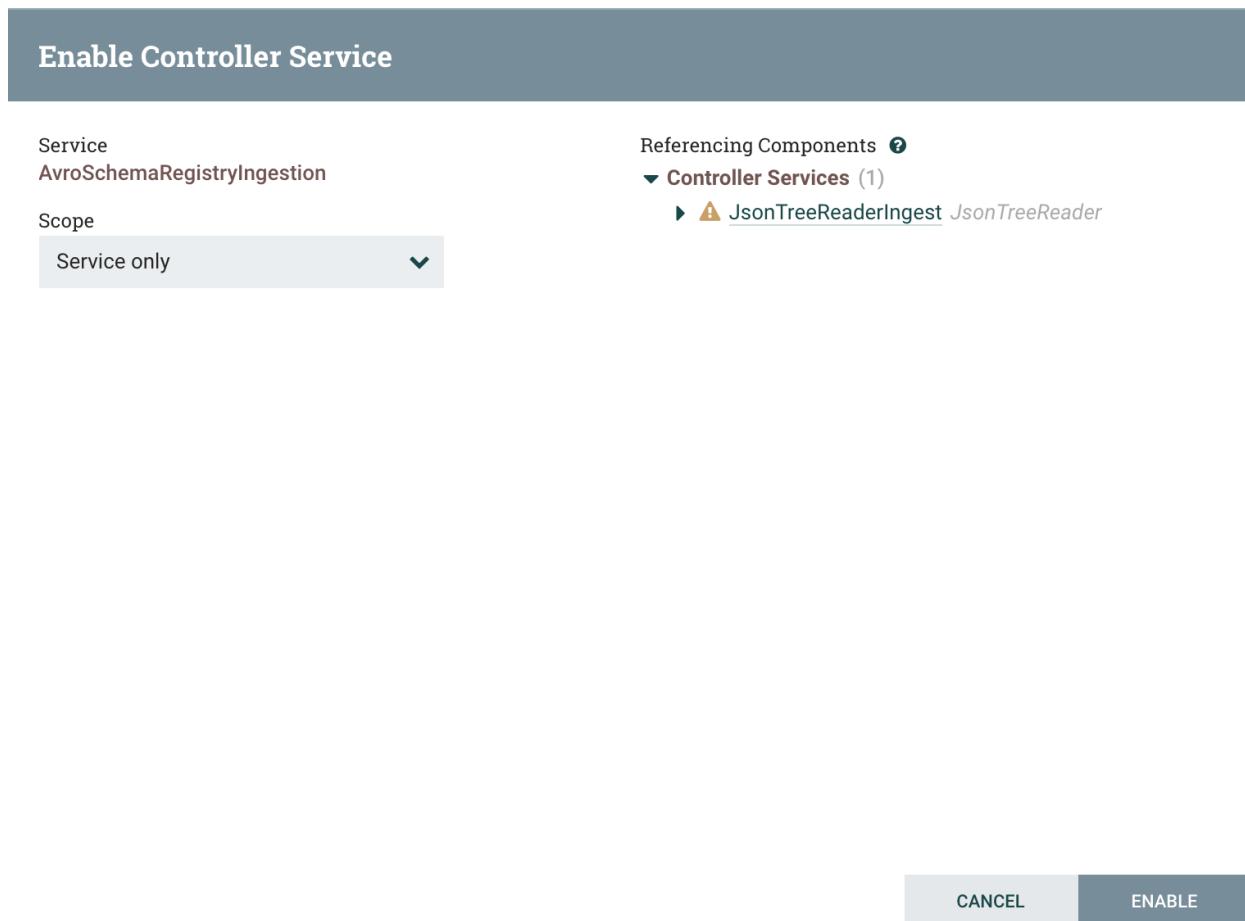
Click **APPLY**

NiFi Flow Configuration

GENERAL CONTROLLER SERVICES

Name	Type	Bundle	State	Scope
AvroSchemaRegistryIngestion	AvroSchemaRegistry 1.8.0	org.apache.nifi - nifi-registry-nar	Disabled	NiFi Flow
JsonTreeReaderIngest	JsonTreeReader 1.8.0	org.apache.nifi - nifi-record-serialization-service...	Invalid	NiFi Flow

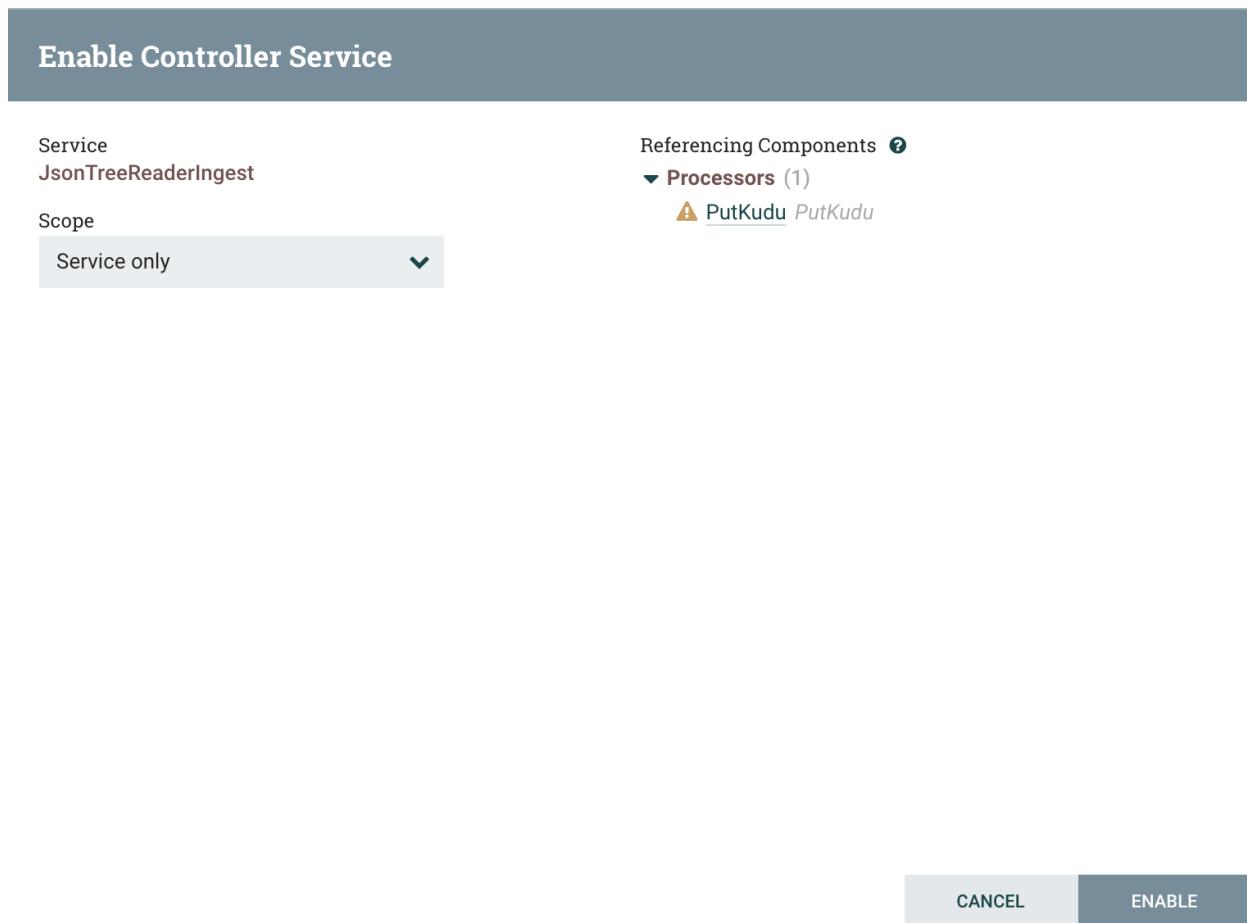
On the right, click the **Enable** icon for the **AvroSchemaRegistryIngestion**



Click **ENABLE** and then **CLOSE**, and the warning icon should disappear



Then enable also the **JsonTreeReaderIngest**



Click **ENABLE** and **CLOSE**



Close the **NiFi Flow Configuration** with the top right X icon

Now drag a connection from the PutKudu to itself

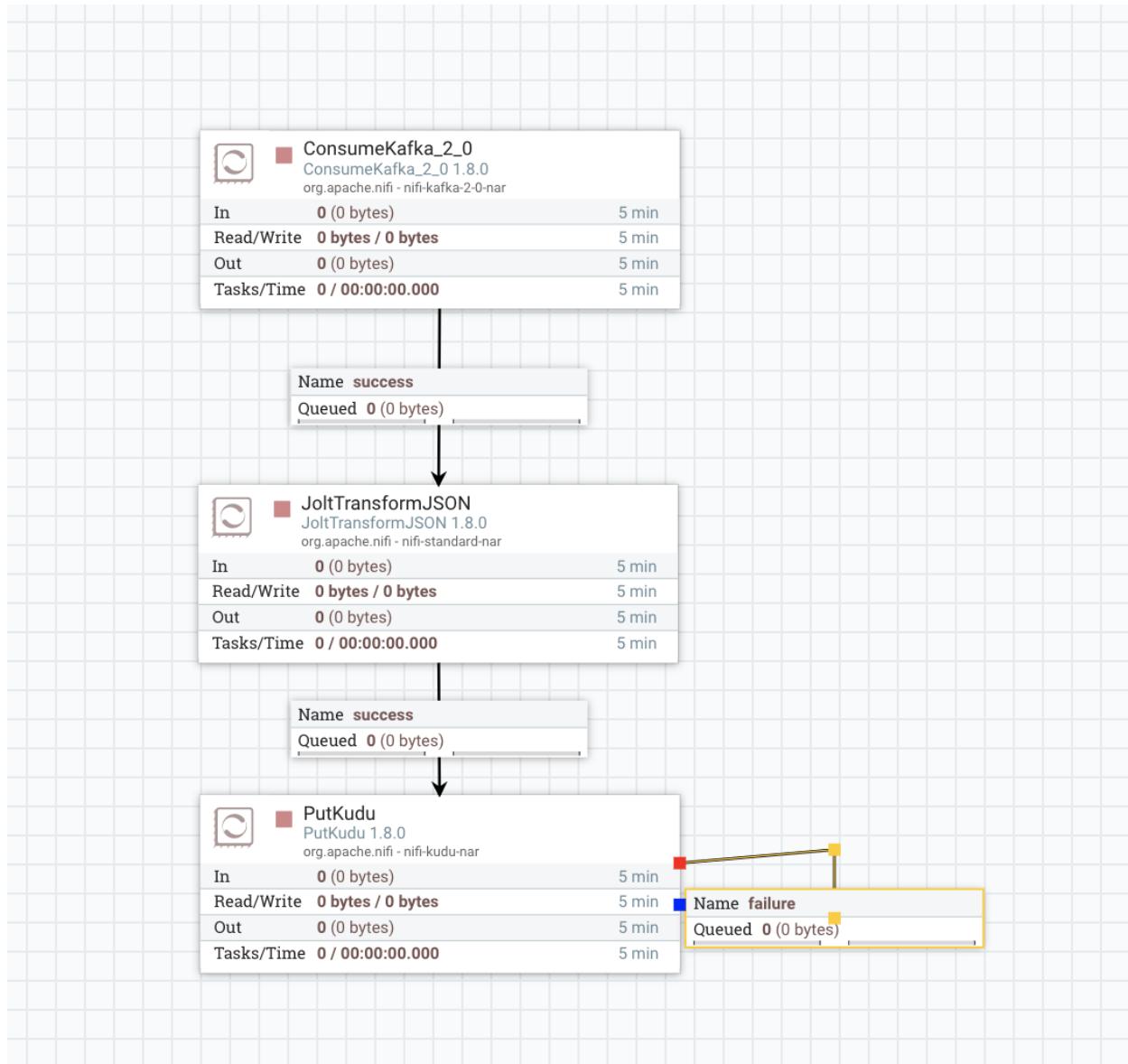
### Create Connection

DETAILS      SETTINGS

From Processor <b>PutKudu</b> PutKudu	To Processor <b>PutKudu</b> PutKudu
Within Group NiFi Flow	Within Group NiFi Flow
For Relationships	
<input checked="" type="checkbox"/> failure	
<input type="checkbox"/> success	

CANCEL      ADD

Then click **ADD**



Remember also to check in the PutKudu step the **Automatically Terminate Relationships** in case of success:

## Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Name: PutKudu     Enabled

Id: a8d4c0cf-0168-1000-308a-04fc4f3ca9aa

Type: PutKudu 1.8.0

Bundle: org.apache.nifi - nifi-kudu-nar

Penalty Duration: 30 sec    Yield Duration: 1 sec

Bulletin Level:

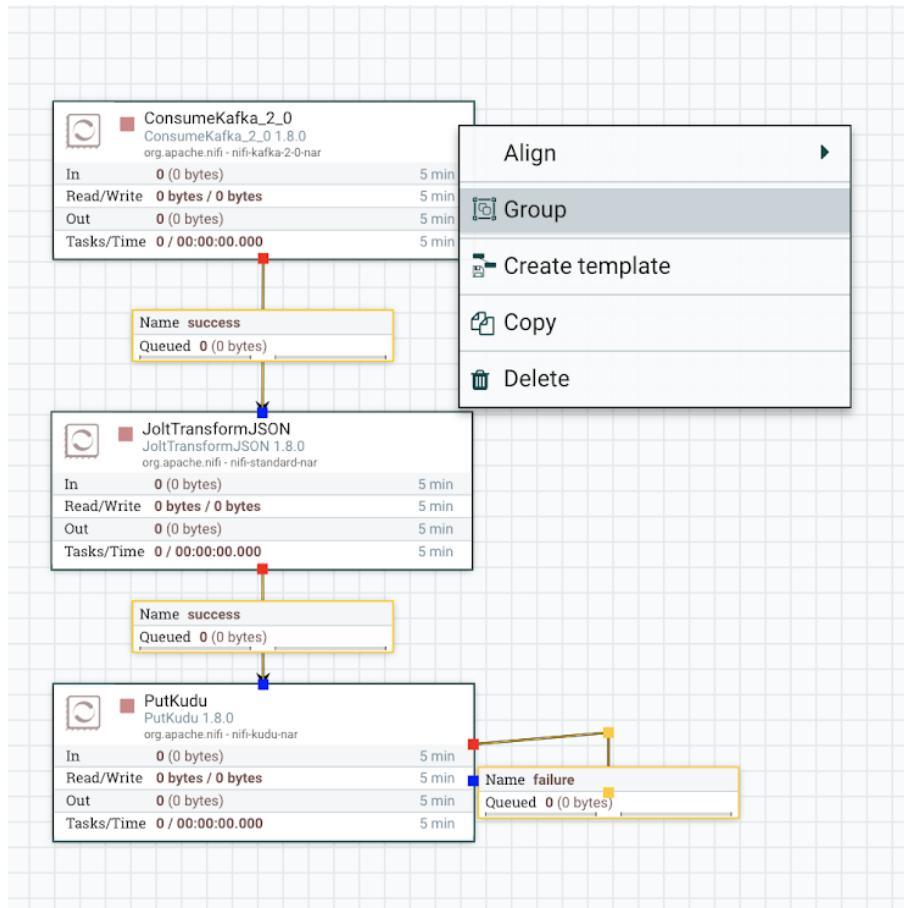
Automatically Terminate Relationships:  failure  
 success

A FlowFile is routed to this relationship if it cannot be sent to Kudu

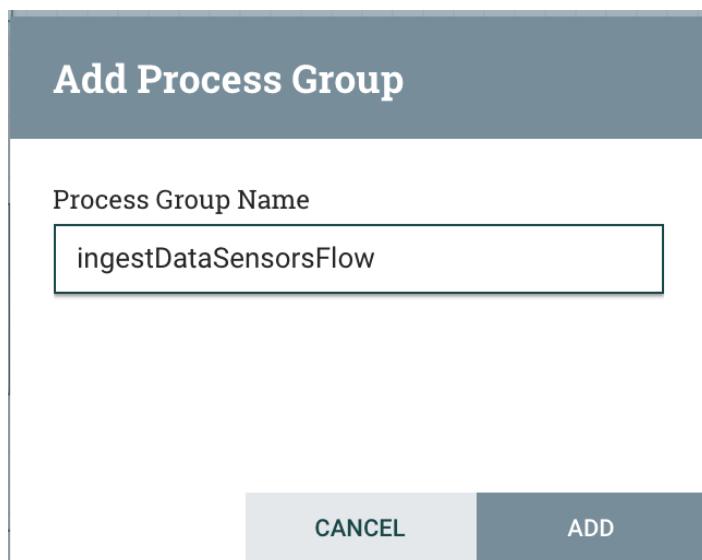
A FlowFile is routed to this relationship after it has been successfully stored in Kudu

CANCEL    APPLY

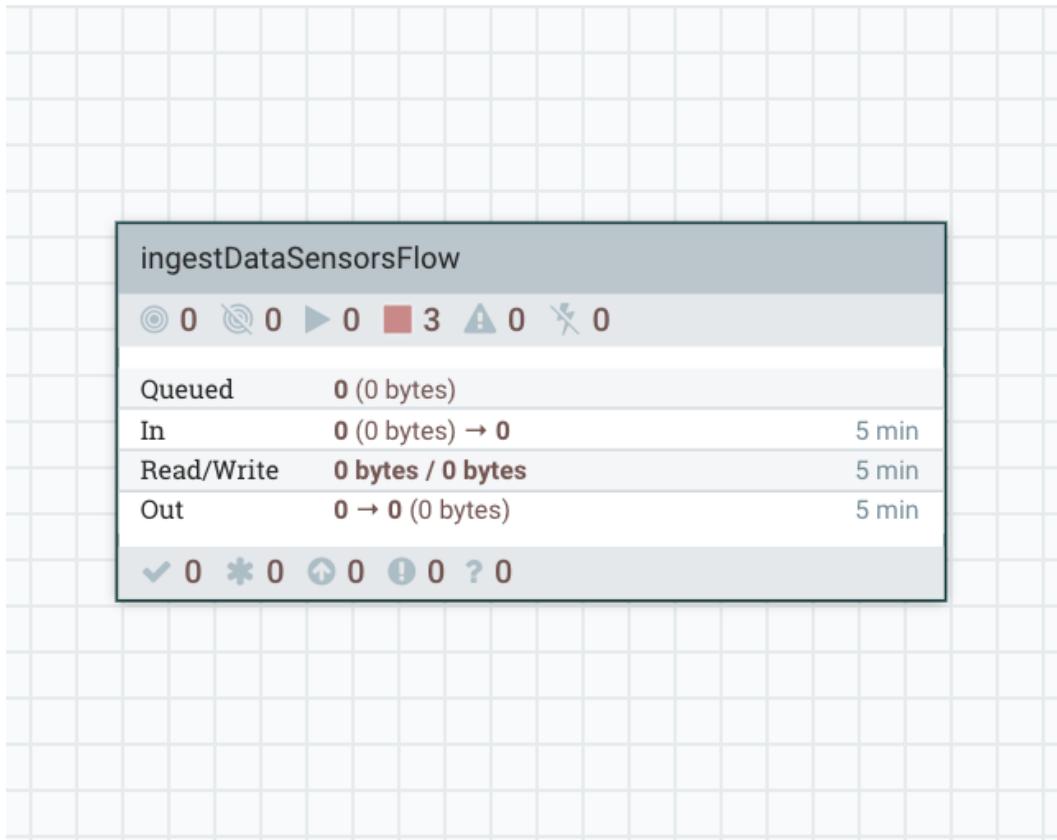
Now the flow is ready to be run, but before to run it, let's create a **Group**.  
Do you remember how to do it? SHIFT hold...



Give it a name, in my case ingestDataSensorsFlow:



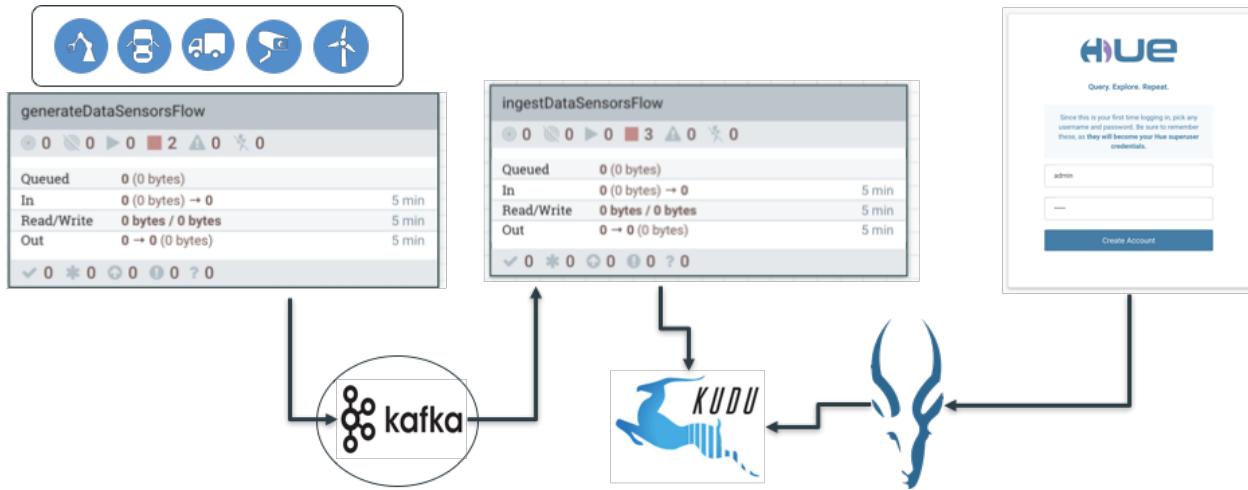
And here our group



Now we are ready to test the end2end flow.

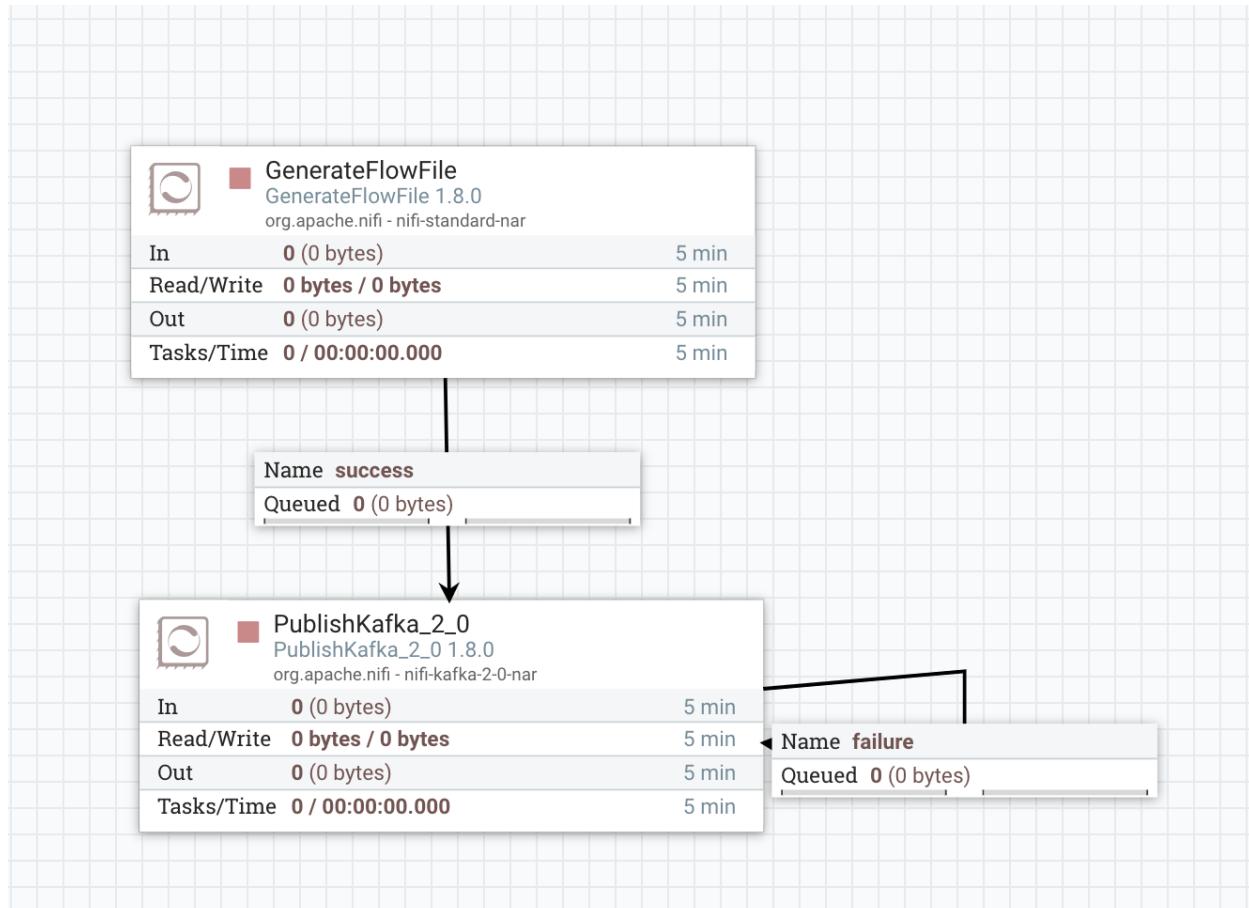
## Lab 6: Run and verify the real-time ingest pipelines

At this point we have built two pipelines in NiFi. The first one feeds into a Kafka topic; the second one consumes from that topic and writes to Kudu:

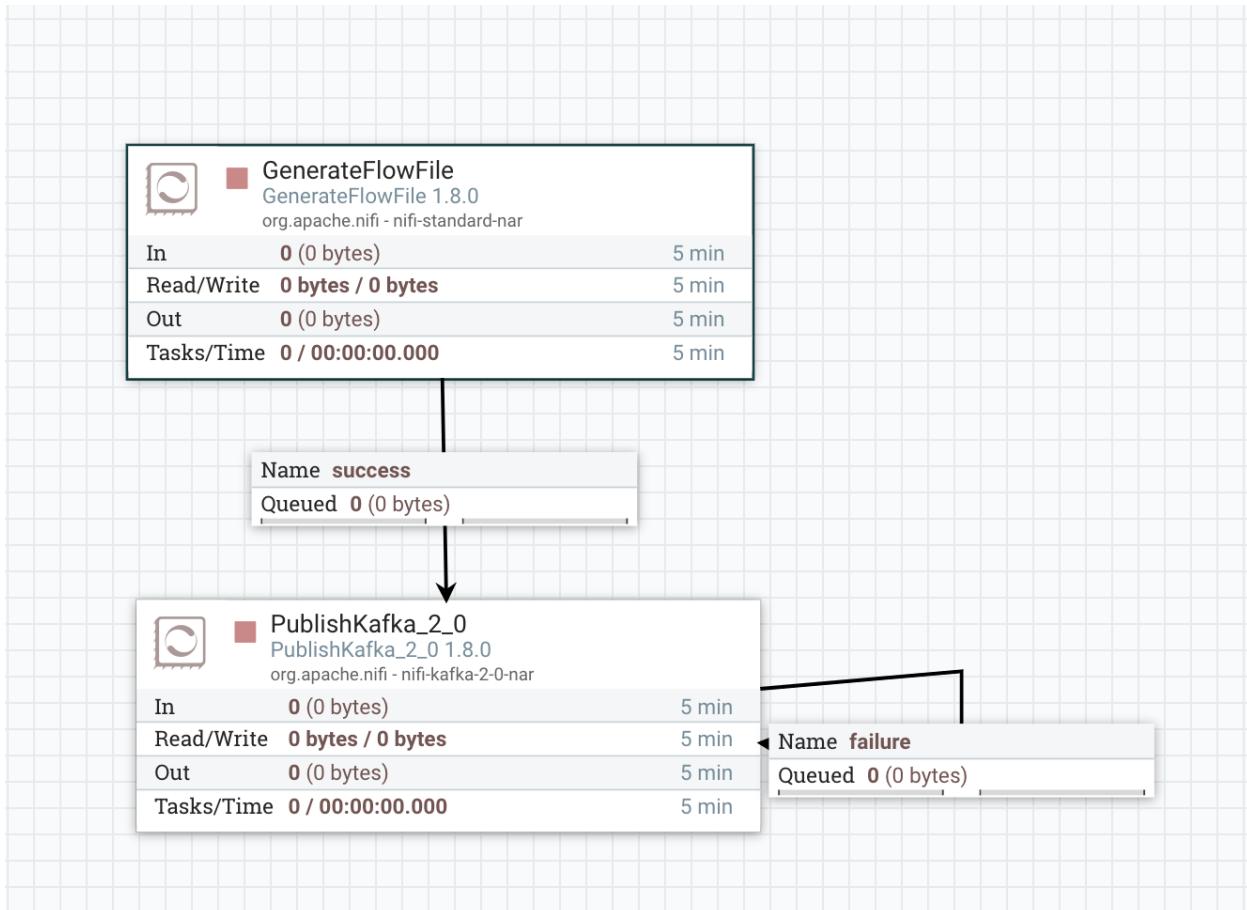


NiFi allows you a fine control on which components in your flows to run, this is very useful when debugging and testing the flows.

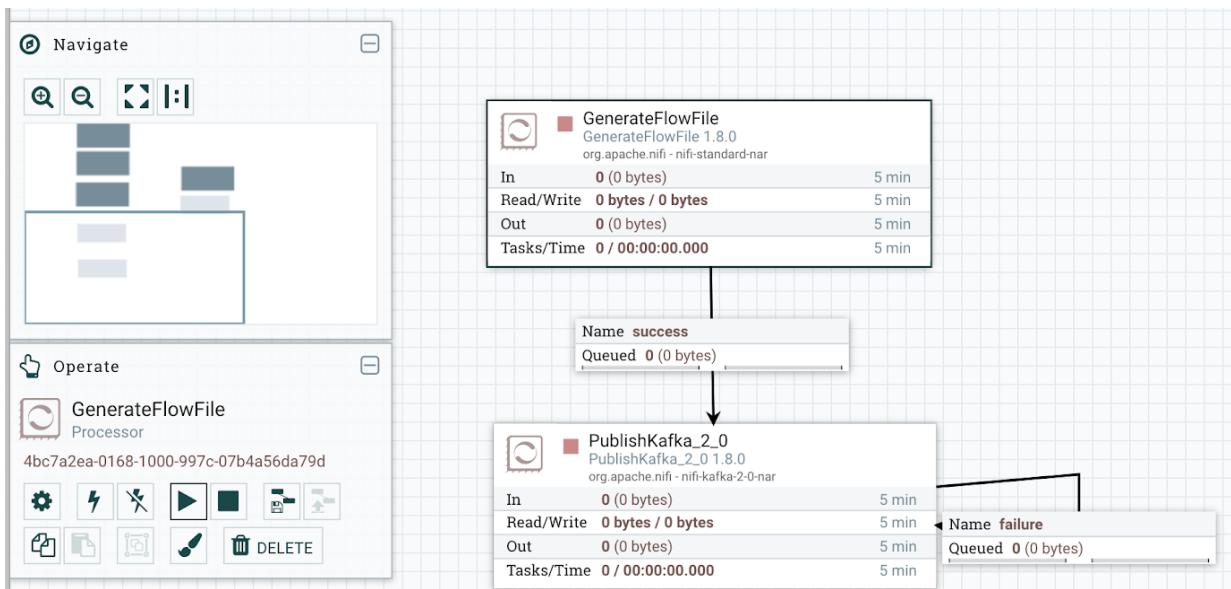
Let's start with the first flow, the one that generate the random sensors readings:



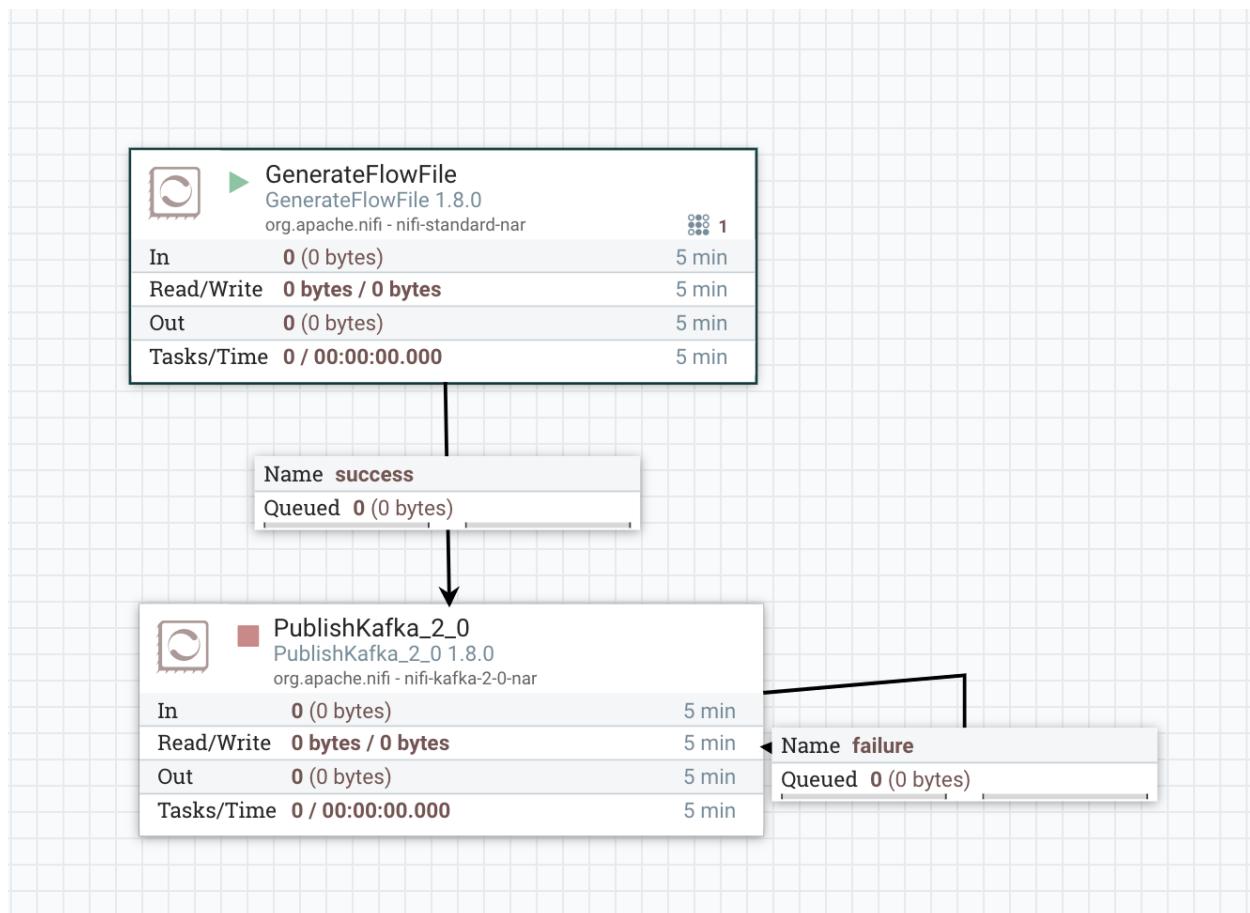
Select the first step:



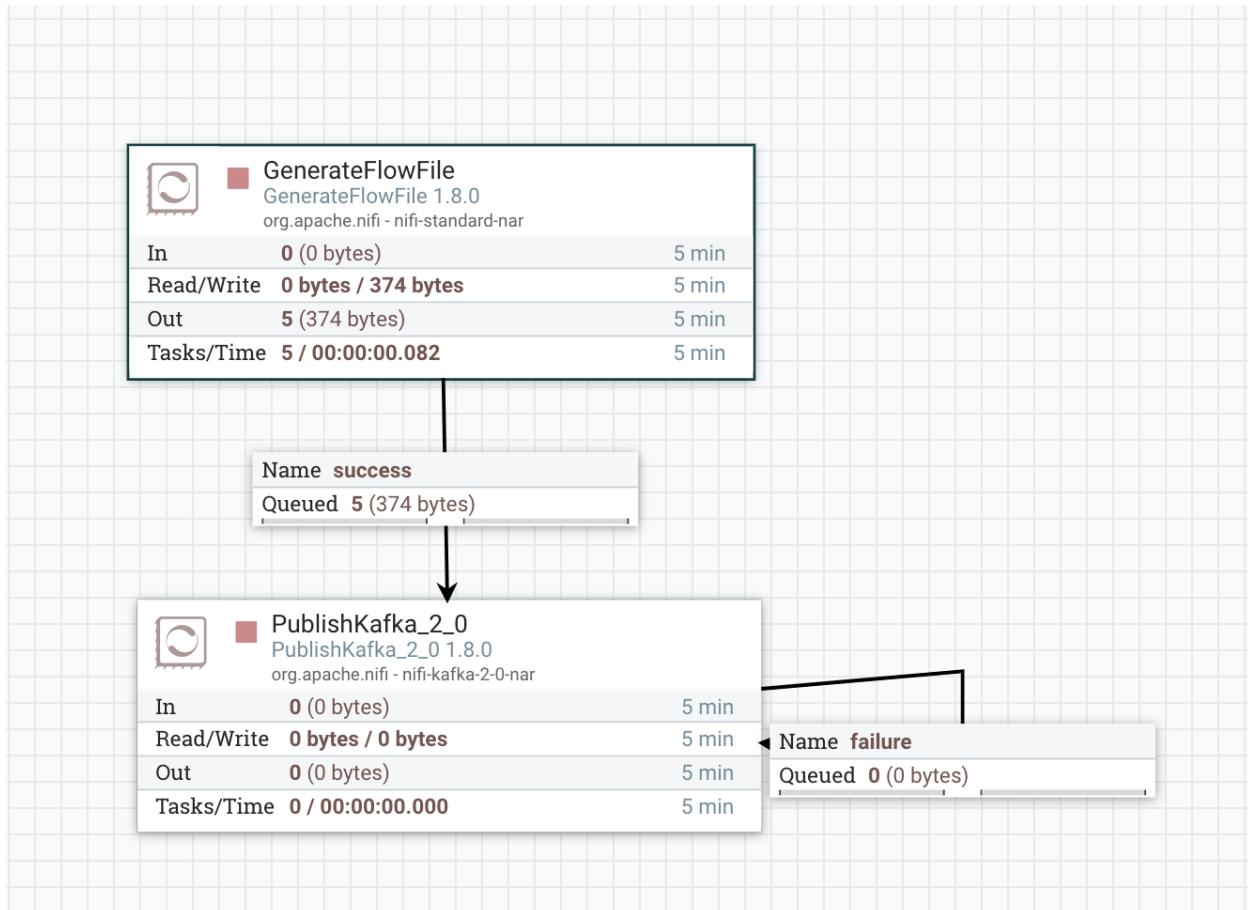
Now, in the **Operate** box on the left click the **Start** icon



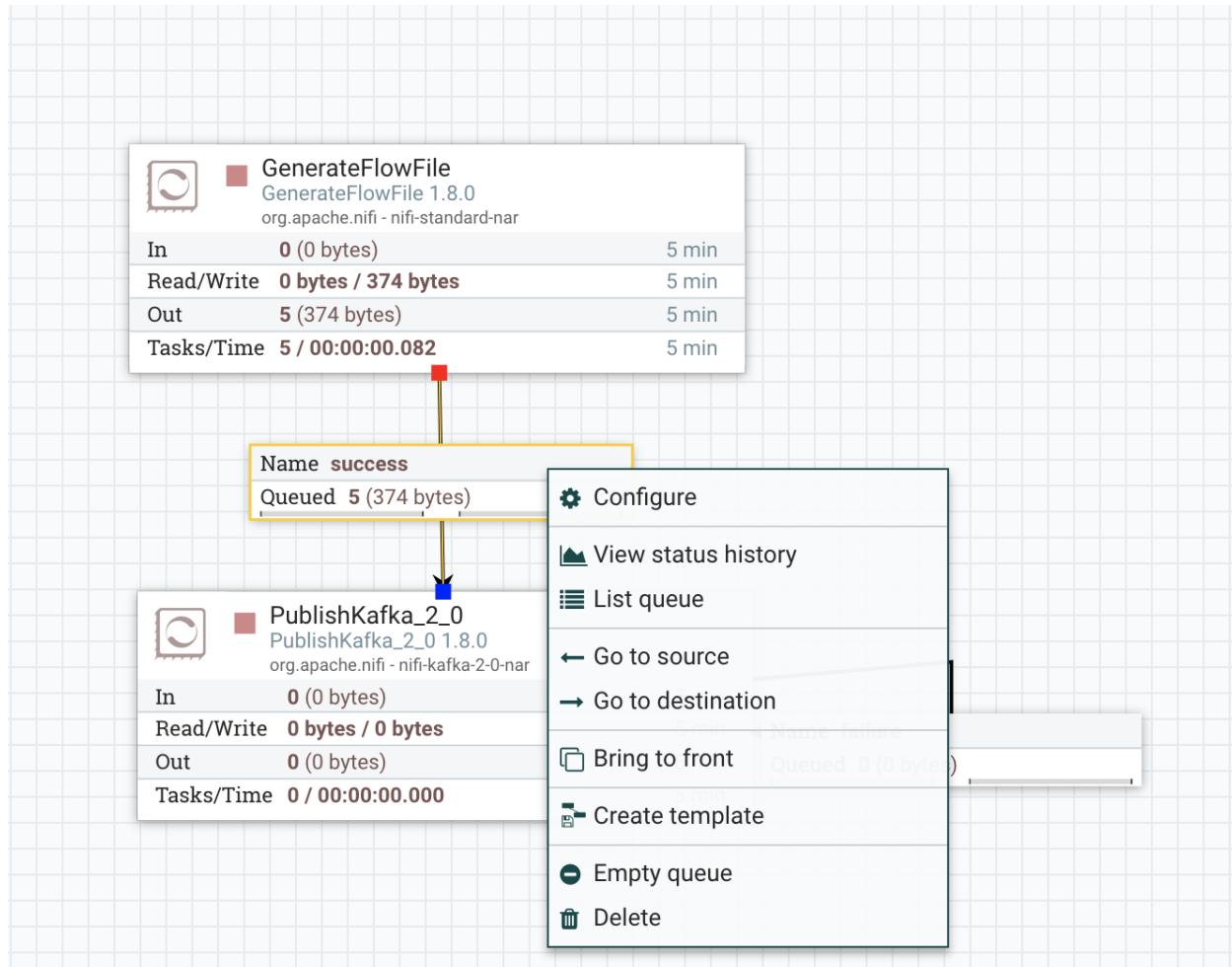
Now you should see the **Run** icon on the top left corner of the **GenerateFlowFile**:



Let it run until you see some messages in the **Queue**, in my case I've 5 messages accumulated in the queue. You can now stop the **GenerateFlowFile** processor.



Select the **Queue** and right click, then select **List queue** option:



You should see the list of all the message awaiting in the queue:

SUCCESS							
Displaying 5 of 5 (374.00 bytes)							
	Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized
1	1	9fae61aa-5ea9-4bf9-bcc2-6d1b725f45...	9fae61aa-5ea9-4bf9-bcc2-6d1b725f45...	74.00 bytes	00:04:48.369	00:04:48.377	
2	2	3afdf51a-d724-4c6a-a1ad-0ed5d92e7...	3afdf51a-d724-4c6a-a1ad-0ed5d92e7...	75.00 bytes	00:04:38.332	00:04:38.332	
3	3	b0ff0afc-5454-4ce6-a8af-0280f95446f8	b0ff0afc-5454-4ce6-a8af-0280f95446f8	75.00 bytes	00:04:28.329	00:04:28.330	
4	4	d2a3df54-7d5f-4722-91f7-35c186679...	d2a3df54-7d5f-4722-91f7-35c186679...	75.00 bytes	00:04:18.325	00:04:18.325	
5	5	d8b6d5d2-0ee3-4768-8224-466698a9...	d8b6d5d2-0ee3-4768-8224-466698a9...	75.00 bytes	00:04:08.323	00:04:08.323	

Choose one message and click on the **Information** icon, in order to view the details

The screenshot shows a 'FlowFile' details view. At the top, there are tabs for 'DETAILS' and 'ATTRIBUTES', with 'ATTRIBUTES' being active. Below this, the 'FlowFile Details' section lists various attributes:

Attribute	Value
UUID	9fae61aa-5ea9-4bf9-bcc2-6d1b725f4530
Filename	9fae61aa-5ea9-4bf9-bcc2-6d1b725f4530
File Size	74 bytes
Queue Position	No value set
Queued Duration	00:08:09.276
Lineage Duration	00:08:09.284
Penalized	No

On the right side, under 'Content Claim', there are fields for Container (Container, default), Section (Section, 1), Identifier (Identifier, 1547460397002-1), Offset (Offset, 0), and Size (Size, 74 bytes). Below these fields are 'DOWNLOAD' and 'VIEW' buttons. In the bottom right corner of the main area is an 'OK' button.

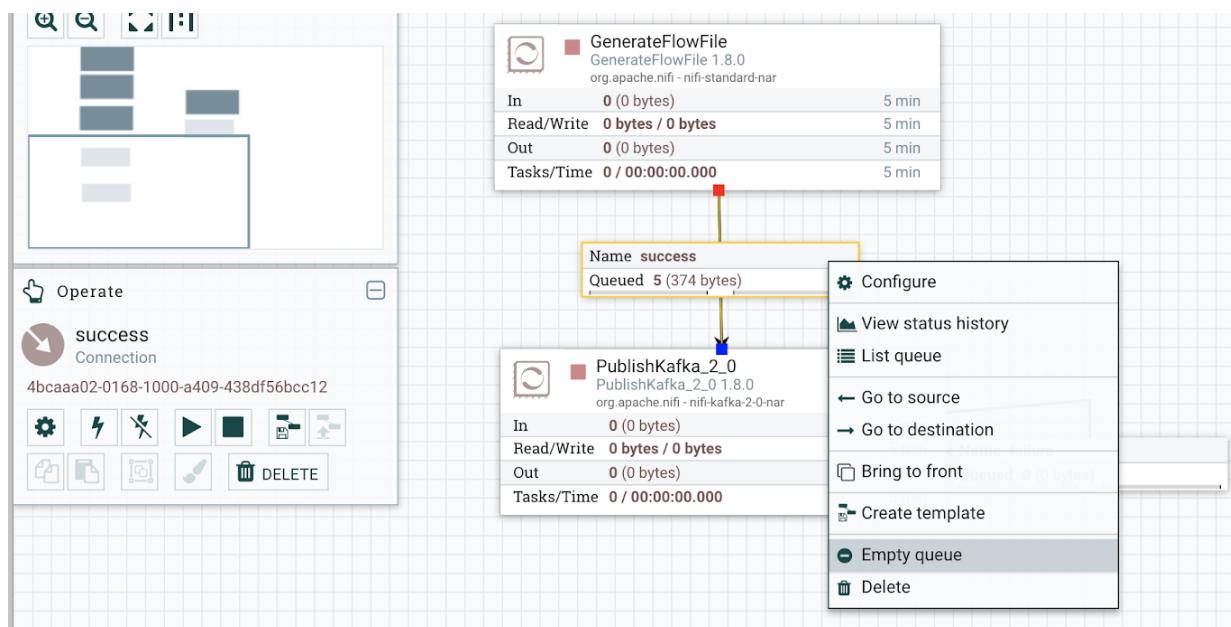
Then click on the **VIEW** icon

The screenshot shows the message content in JSON format. The 'View as' dropdown is set to 'original'. The message is displayed as:

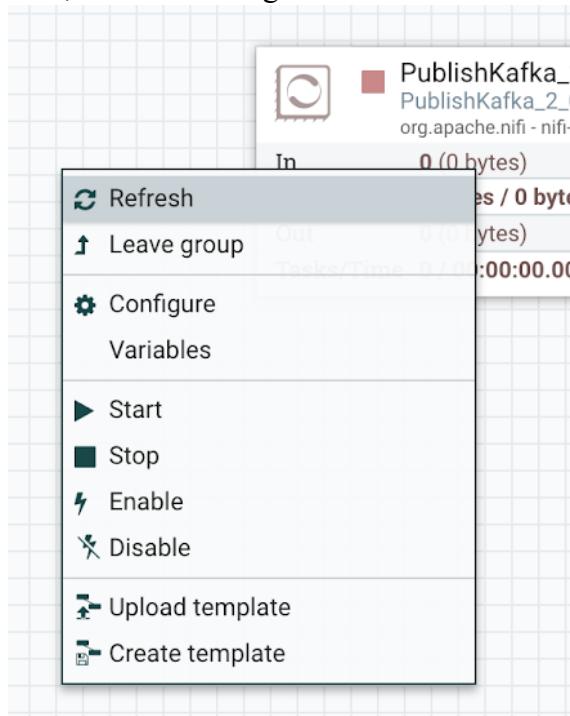
```
1 {  
2   "sensorID": 3709,  
3   "sensorValue": 93,  
4   "sensorTimestamp": 1547460396996000  
5 }  
6
```

This is exactly the message format that we wanted to generate. Double check other messages in order to see if they are conform to the same format.

We could empty the queue **but do not do it**; empty the queue maybe useful when debugging your flow you have realized that the messages are not corrected for the downstream steps:



We want to see if the kafka components are working fine. So in order to do that go to the second flow, the one that ingest the data. In order to do it you can click in an empty point in the canvas:

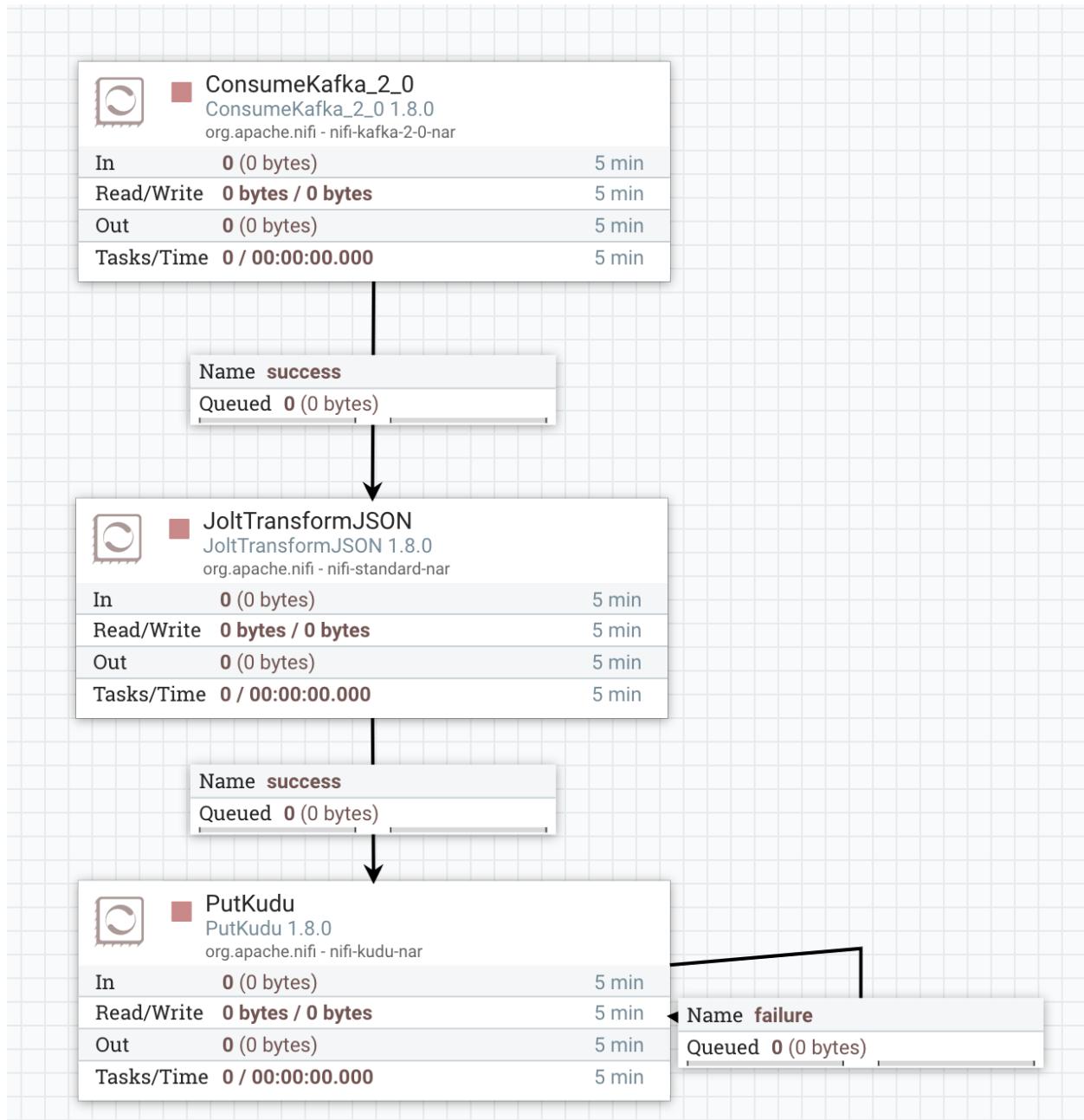


And click **Leave group**

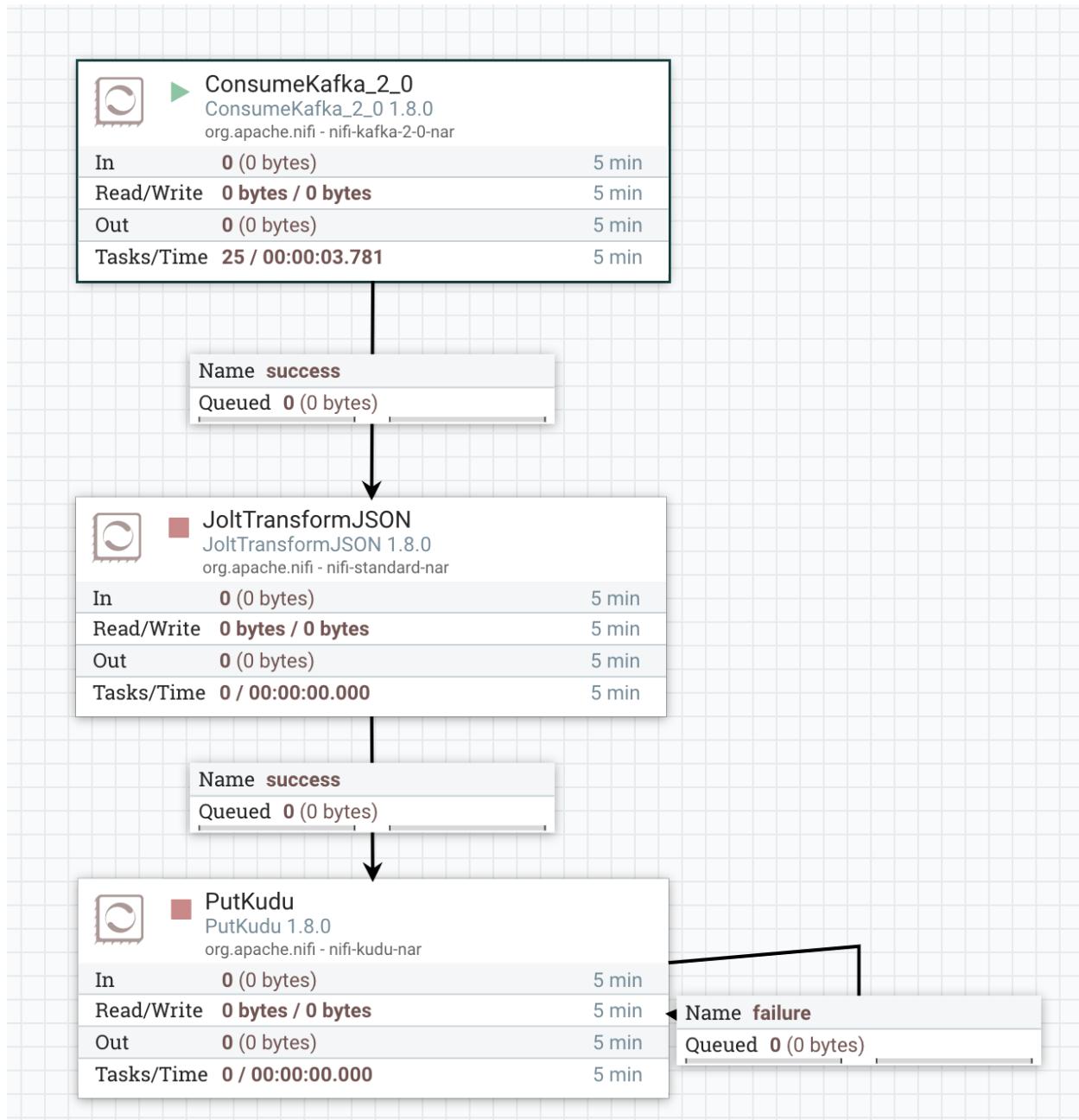
Alternatively, you can also click the NiFi link at the bottom of the web UI:



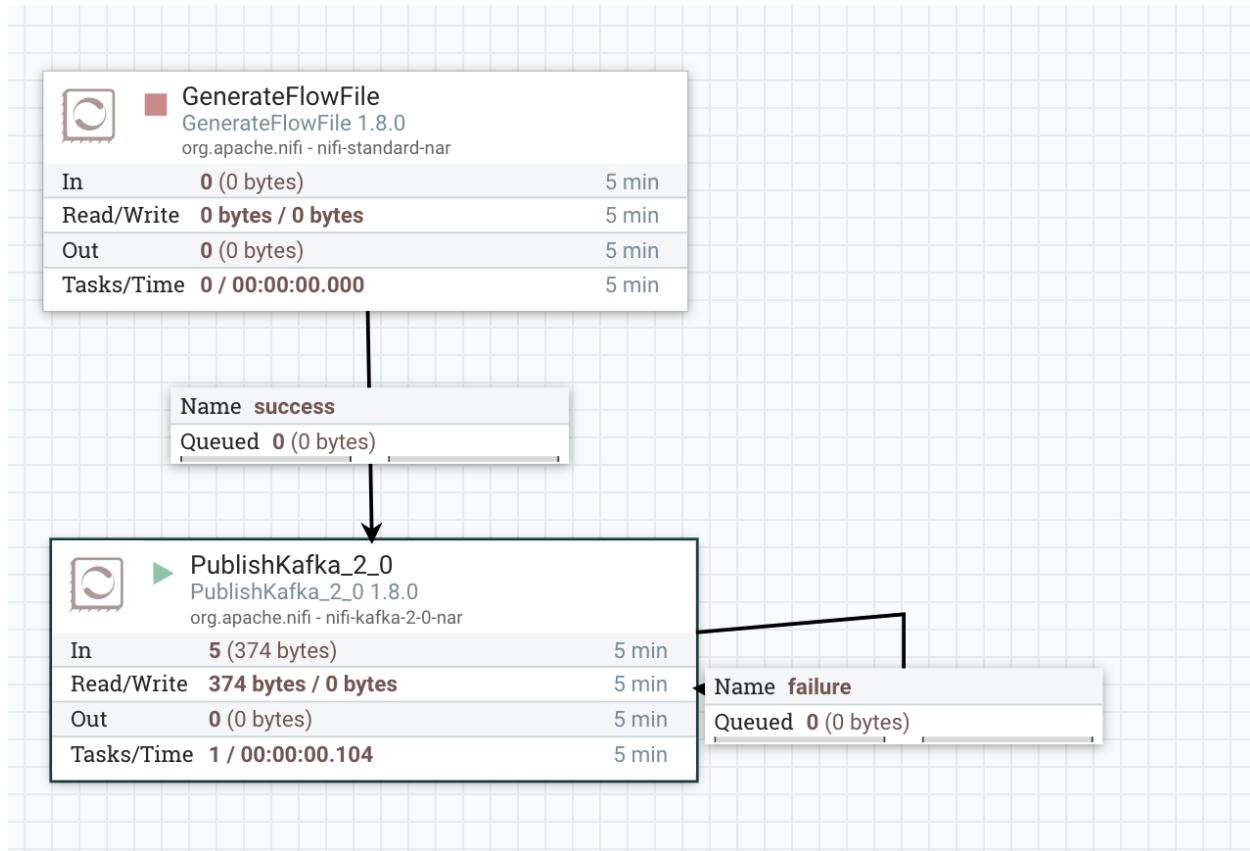
And then go to the *ingestDataSensorsFlow*:



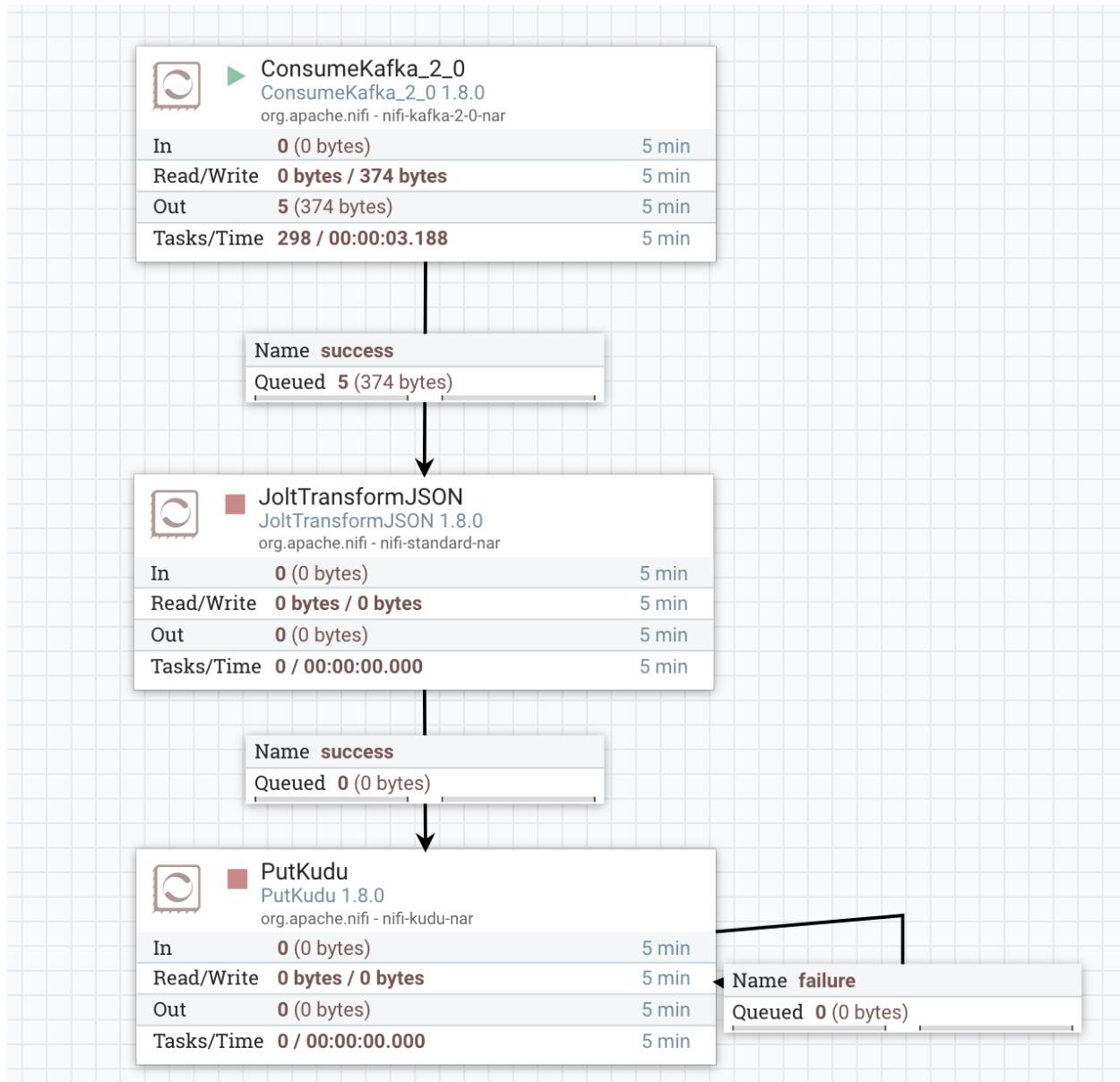
And run the **ConsumeKafka** processor:



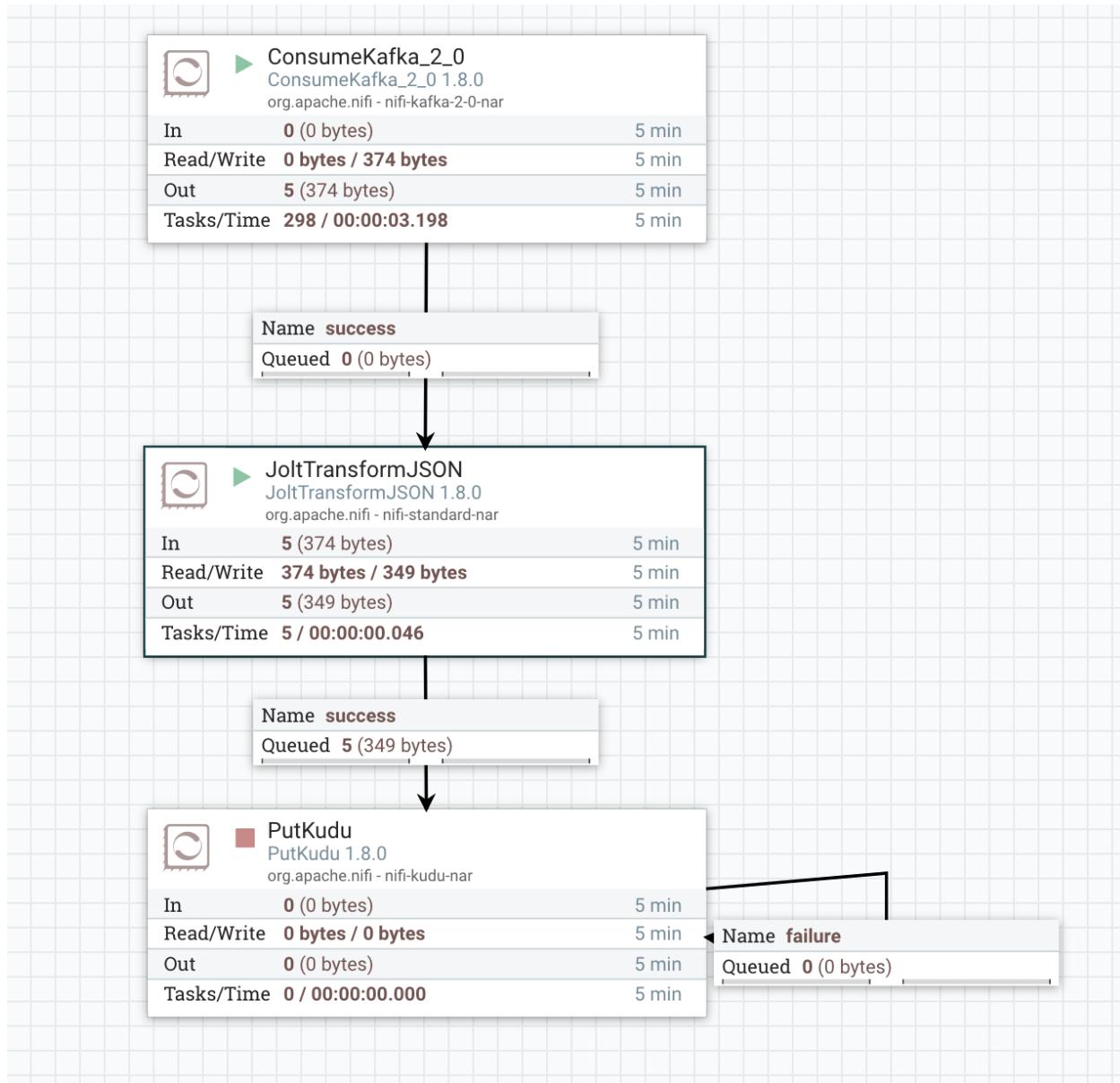
It does not consume messages because the **PublishKafka** component on the first flow is still in **Stop** state. Go back to the first flow and run the **PublishKafka**



As you can see the Queue now is empty. Go back to the ingest flow:



As you can notice, now we have 5 messages in this queue. Check the content of the messages.  
Run now the **JoltTransformJSON**:



Now the 5 messages are in the queue in front of the **PutKudu** steps. Check the content of some message to see the transformation worked properly:

View as: original ▾

```

1 { "sensorid":3709, "sensorvalue":93, "sensortimestamp":1547460396996000}

```

The fields are all in lowercase, as we were expected.

You can also check the provenance feature of NiFi.

Go back to the queue list, and click on the Provenance icon below:

The source of this queue is currently running. This listing may no longer be accurate.						
Queued Duration		Lineage Duration		Penalized		
	00:02:15.693		00:02:39.245			
	00:02:15.690		00:02:39.243			
	00:02:15.688		00:02:39.242			
	00:02:15.686		00:02:39.242			

### NiFi Data Provenance

Displaying 3 of 3  
Oldest event available: 01/31/2019 13:48:51 UTC

Filter		by component name ▾					
Date/Time	Type	FlowFileUuid	Size	Component Name ▾	Component Type		
01/31/2019 14:02:43.753 UTC	DOWNLOAD	e94624f8-715a-44ed-90...	70 bytes	No value set	NiFi Flow		
01/31/2019 14:01:29.464 UTC	RECEIVE	e94624f8-715a-44ed-90...	70 bytes	ConsumeKafka_2_0	ConsumeKafka_2_0		
01/31/2019 14:01:53.020 UTC	CONTENT_MODIFIED	e94624f8-715a-44ed-90...	70 bytes	JoltTransformJSON	JoltTransformJSON		

Switch to HUE interface, <master IP>:8888 (in my case <http://flamb2-mn0.westeurope.cloudapp.azure.com:8888>), and run the following query to check that the sensor table is empty:

```
SELECT count(*) FROM sensor_table;
```

The screenshot shows the Cloudera Impala Query Editor interface. At the top, there's a header with the title 'Impala' and buttons for 'Add a name...' and 'Add a description...'. Below the header, a code editor window contains the following SQL query:

```
1| SELECT count(*) FROM sensor_table;
```

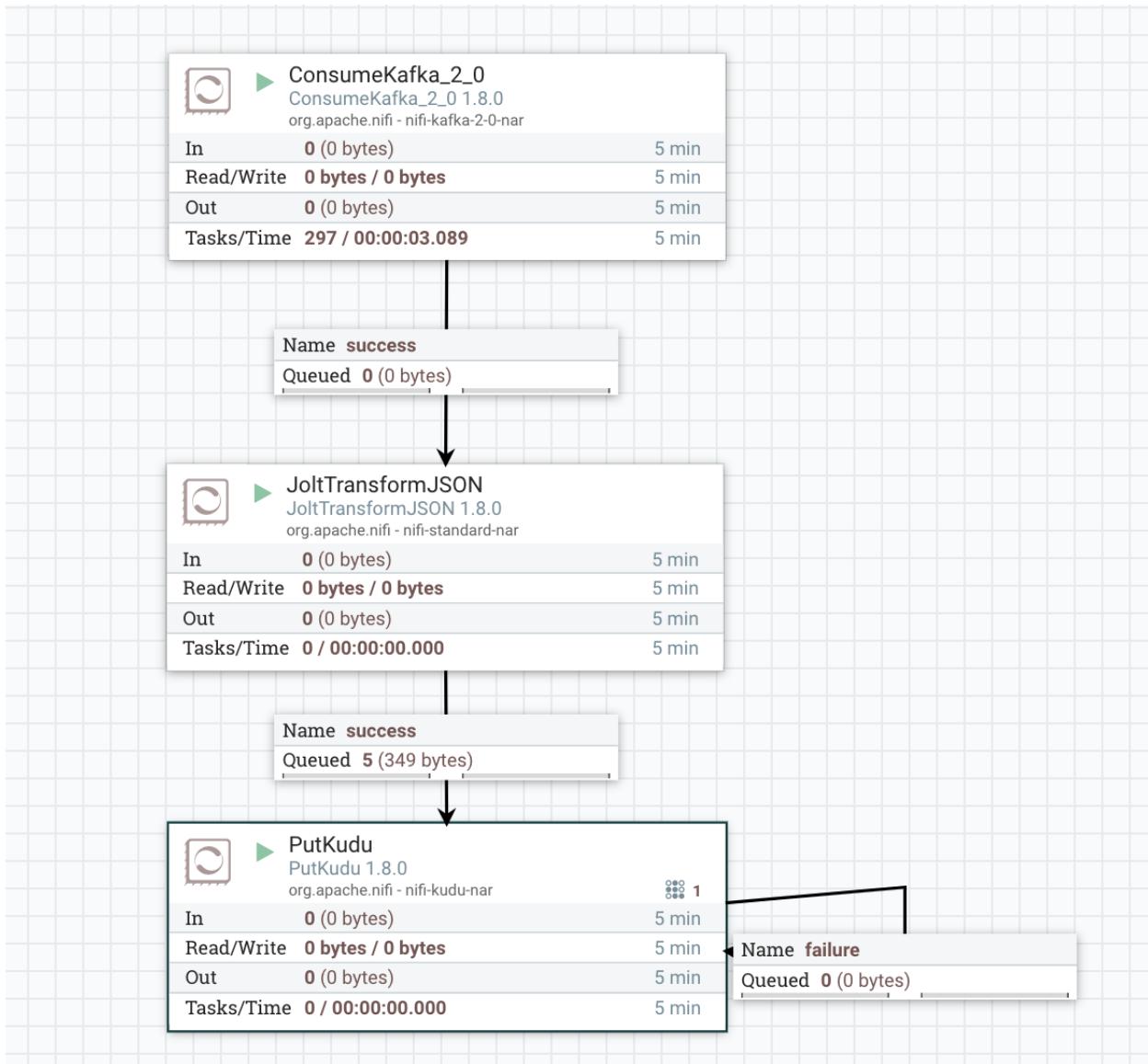
Below the code editor are two small icons: a play button and a map pin.

Underneath the code editor is a progress bar indicating 'Query 9b4cf3680d4076ff:e78db59000000000 100% Complete (16 out of 16)'.

At the bottom of the interface, there are three tabs: 'Query History' (with a magnifying glass icon), 'Saved Queries' (with a magnifying glass icon), and 'Results (1)' (which is active, indicated by a blue underline and a magnifying glass icon). The 'Results (1)' tab displays the following table:

count(*)
1 0

Now we can run the last step, the **PutKudu** processor:



Run again the SELECT query:

The screenshot shows the Cloudera Impala Query Editor interface. At the top, there's a header with the Cloudera logo, the word "Impala", and buttons to "Add a name..." and "Add a description...". Below the header, a query editor window contains the following text:

```
1| SELECT count(*) FROM sensor_table;
```

Below the editor are two small icons: a play button and a folder icon with a downward arrow.

Underneath the editor is a progress bar indicating "Query 624cbe90c9f81cccd:e470c62f00000000 100% Complete (16 out of 16)".

At the bottom of the interface, there are three tabs: "Query History" (with a search icon), "Saved Queries" (with a search icon), and "Results (1)" (with a search icon). The "Results (1)" tab is currently selected.

The results section displays the output of the query:

count(*)
1 5

There are also small icons for a grid and a chart next to the results table.

You can see that we have 5 messages. Check the content of these messages with the following statement:

```
SELECT * FROM sensor_table;
```

The screenshot shows the Cloudera Impala interface. At the top, there are tabs for 'Impala' and 'Add a name...', and icons for saving and running queries. Below the tabs, the status bar shows '0.54s default text ?'. The main area contains a code editor with two queries:

```

1 SELECT count(*) FROM sensor_table;
2
3
4 SELECT * FROM sensor_table;

```

Below the code editor is a progress bar indicating 'Query fe47a34a86f53ac3:c200bea500000000 100% Complete (16 out of 16)'. The results section shows a table with three columns: 'sensorid', 'sensorvalue', and 'sensortimestamp'. The data is as follows:

	sensorid	sensorvalue	sensortimestamp
1	3929	123	2019-01-14 10:07:17.052000000
2	7057	227	2019-01-14 10:07:07.050000000
3	8891	204	2019-01-14 10:06:57.045000000
4	3709	93	2019-01-14 10:06:36.996000000
5	6546	121	2019-01-14 10:06:47.043000000

Now If you want you can run all the components of the two flows in parallel; once done then stop the two NiFi pipelines with the **Stop** icon.

### Hint

In order to generate high volume of the random sensor readings, change the **Run Schedule** field from 10 sec to a smaller value (the value of 0 sec means that the Processor should run as often as possible) or increase the number of **Concurrent Tasks**; in the **SCHEDULING** tab of the **GenerateFlowFile** component:

### Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Scheduling Strategy ?  
Timer driven

Concurrent Tasks ?  
1

Execution ?  
All nodes

Run Duration ?  
0ms 25ms 50ms 100ms 250ms 500ms 1s 2s  
Lower latency      Higher throughput

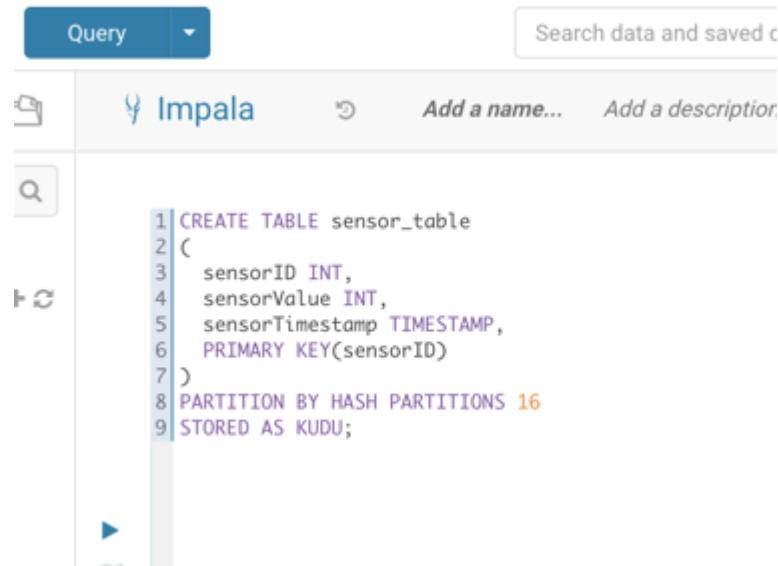
Run Schedule ?  
10 sec

CANCEL    APPLY

## Lab 7: Improve Kudu table partitioning strategies

### Partition strategy improvement #1

In our sample example we have used a simple Kudu table with a simple partitioning schema:



```
1 CREATE TABLE sensor_table
2 (
3     sensorID INT,
4     sensorValue INT,
5     sensorTimestamp TIMESTAMP,
6     PRIMARY KEY(sensorID)
7 )
8 PARTITION BY HASH PARTITIONS 16
9 STORED AS KUDU;
```

This schema does not take into consideration the timestamp of the reading, something that you want to consider in order to improve the performance of your solution. In this exercise we are now going to introduce a new partitioning schema considering also the timestamp field.

In Hue create a new Kudu table as following **but replace the dates in the partitions section starting with the day of today**.

```
CREATE TABLE improve_sensor_table
(
    sensorID INT,
    sensorTimestamp TIMESTAMP,
    sensorValue INT,
    PRIMARY KEY (sensorID, sensorTimestamp)
)
PARTITION BY HASH PARTITIONS 4,
RANGE (sensorTimestamp)
(
    PARTITION ('2019-02-01 00:00:00') <= VALUES < ('2019-02-02 00:00:00'),
    PARTITION ('2019-02-02 00:00:00') <= VALUES < ('2019-02-03 00:00:00')
)
STORED AS KUDU;
```

```

16 CREATE TABLE improve_sensor_table
17 (
18     sensorID INT,
19     sensorTimestamp TIMESTAMP,
20     sensorValue INT,
21     PRIMARY KEY (sensorID, sensorTimestamp)
22 )
23 PARTITION BY HASH PARTITIONS 4,
24 RANGE (sensorTimestamp)
25 (
26     PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
27     PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00')
28 )
29 )
30 STORED AS KUDU;
31

```

Success.

Query History    Saved Queries

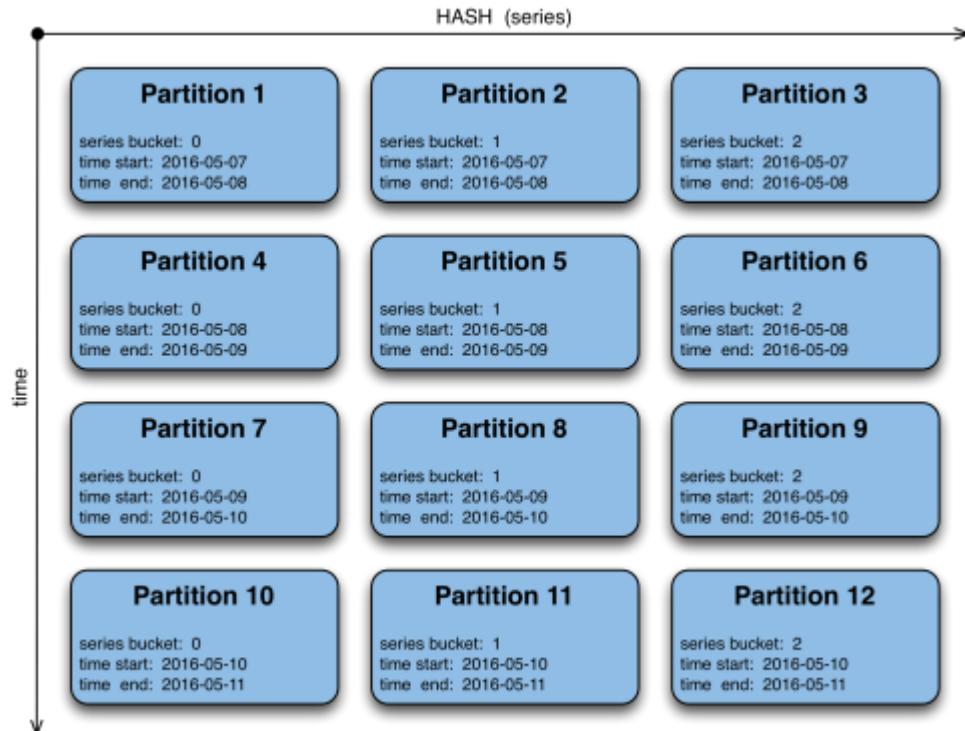
un minuto fa ✓

```

CREATE TABLE improve_sensor_table ( sensorID INT, sensorTimestamp TIMESTAMP, sensorValue INT,
PRIMARY KEY (sensorID, sensorTimestamp) ) PARTITION BY HASH PARTITIONS 4, RANGE (sensorTimestamp) (
PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00') ) STORED AS KUDU

```

Here we are using a nested partitioning strategy, with the hash partitioning for the sensorID and with a range partitioning for the timestamp. Conceptually it looks like this:



## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

partitions after the creation of the table. Let's add new partition with the following SQL statement (**remember always to change the dates in the partition**)

```
ALTER TABLE improve_sensor_table ADD RANGE PARTITION ('2019-02-03 00:00:00') <= VALUES < ('2019-02-04 00:00:00');
```

```
33 | ALTER TABLE improve_sensor_table ADD RANGE PARTITION ('2018-10-10 00:00:00') <= VALUES < ('2018-10-11 00:00:00');

Success.

Query History | Saved Queries
alcuni secondi fa | ALTER TABLE improve_sensor_table ADD RANGE PARTITION ('2018-10-10 00:00:00') <= VALUES < ('2018-10-11 00:00:00')
```

Now that we have created a new table and added additional partitions, let's switch to the Kudu Master web interface to check the new table and partitions.

HINT – port 8051 is not yet open. Use your Azure portal to open that port.

Go to the url <your public Ip>:8051; then click on **Tables**:

Table Name	Table Id	State
impa...::default.improve_sensor_table	9616bf1dfcb84f99b9cc93750da0ad96	Running
impa...::default.sensor_table	e915223e0b4f485e863f6aa9fde8f66c	Running

Click on the **improve\_sensor\_table**:

## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

### Schema

Column	ID	Type	Encoding	Compression	Read default	Write default
sensorid	0	int32 NOT NULL	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-
sensortimestamp	1	unixtime_micros NOT NULL	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-
sensorvalue	2	int32 Nullable	AUTO_ENCODING	DEFAULT_COMPRESSION	-	-

### Partition Schema

```
HASH (sensorid, sensortimestamp) PARTITIONS 4,  
RANGE (sensortimestamp) (  
    PARTITION 2018-09-18T00:00:00.000000Z <= VALUES < 2018-09-19T00:00:00.000000Z,  
    PARTITION 2018-09-19T00:00:00.000000Z <= VALUES < 2018-09-20T00:00:00.000000Z,  
    PARTITION 2018-09-20T00:00:00.000000Z <= VALUES < 2018-09-21T00:00:00.000000Z  
)
```

### Tablets

#### Summary

State	Count	Percentage
RUNNING	12	100.00

Where the number of Tablets is  $12 = \# \text{ Hash partition} \times \# \text{ time Range}$

Go back to the NiFi ingest pipeline and change the table name with this new table that you have just created and run again the two flows (the ingest and the Connected Things).

Check in Hue and note (if you have left the flows running for a while - look the Hint in previous section) that now we have more than 10,000 rows because now the key is not just sensorID but sensorID and timestamp:

```
SELECT count(*) FROM improve_sensor_table;
```

Query History



Saved Queries



Results (1)



count(\*)



1 50816



```
SELECT * FROM improve_sensor_table WHERE sensorid=10000;
```

The screenshot shows the Cloudera Hue interface with a query results page. At the top, a code editor shows a SQL query: `43| SELECT * FROM test_sensor_table WHERE sensorid=10000;`. Below the code editor, a status bar indicates "Query d5432b189214f0fa:83da202a00000000 100% Complete (12 out of 12)". To the right, a URL `d5432b189214f0fa:83da20` is highlighted. The main area displays a table with six rows of data:

	sensorid	sensortimestamp	sensorvalue
1	10000	2018-09-18 13:30:03.097000000	73
2	10000	2018-09-18 13:30:34.932000000	178
3	10000	2018-09-18 13:30:36.973000000	64
4	10000	2018-09-18 13:30:43.138000000	36
5	10000	2018-09-18 13:30:44.155000000	212
6	10000	2018-09-18 13:30:23.659000000	179

Stop the two NiFi pipelines.

## Partition strategy improvement #2

Tablets store the rows in primary key order, so if we put the timestamp at the front of the primary key, then the tablets are mostly being appended to the end of, rather than inserted all through the middle. The reason this is mostly an append is because new rows have a higher timestamp than most of the rows that already exist.

Appends operation are better because Kudu does not need to spend time compacting new records into existing tablets row sets, and because Kudu does not need to cache metadata about every part of every tablet (if this cache starts thrashing then write performance will crater).

So, go back to Hue interface and create a new table with a better partition strategy for dealing with time-series data like those in our case (remember always to change the dates in the partitions):

```
CREATE TABLE improve2_sensor_table
(
    sensorTimestamp TIMESTAMP,
    sensorID INT,
    sensorValue INT,
    PRIMARY KEY (sensorTimestamp, sensorID)
)
PARTITION BY HASH PARTITIONS 4,
RANGE (sensorTimestamp)
(
    PARTITION ('2019-02-01 00:00:00') <= VALUES < ('2019-02-02 00:00:00'),
    PARTITION ('2019-02-02 00:00:00') <= VALUES < ('2019-02-03 00:00:00')
)
STORED AS KUDU;
```

```
43 CREATE TABLE improve2_sensor_table
44 (
45     sensorID INT,
46     sensorTimestamp TIMESTAMP,
47     sensorValue INT,
48     PRIMARY KEY (sensorTimestamp, sensorID)
49 )
50 PARTITION BY HASH PARTITIONS 4,
51 RANGE (sensorTimestamp)
52 (
53     PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'),
54     PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00')
55 )
56 STORED AS KUDU;
```

Success.

Query History    Saved Queries

```
alcuni secondi fa ✓ CREATE TABLE improve2_sensor_table ( sensorID INT, sensorTimestamp TIMESTAMP, sensorValue INT, PRIMARY KEY (sensorTimestamp, sensorID) ) PARTITION BY HASH PARTITIONS 4, RANGE (sensorTimestamp) ( PARTITION ('2018-10-08 00:00:00') <= VALUES < ('2018-10-09 00:00:00'), PARTITION ('2018-10-09 00:00:00') <= VALUES < ('2018-10-10 00:00:00') ) STORED AS KUDU
```

Go back to the NiFi ingest pipeline and change the table name with this new table that you have just created, run again the two pipelines (the ingest and the Connected Things).

Once you have and verify in Hue that this new table is being populated, then you can stop the two StreamSets pipelines.

## Simple Queries

Let's understand more the values ingested in Kudu. First we need to identify the sensor that has generated more values, in order to do to type in Hue the following query:

```
select count(*) as Y, sensorid
from improve2_sensor_table
group by sensorid
order by Y DESC
limit 10;
```

## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

```
59 | select count(*) as Y, sensorid
60 | from improve2_sensor_table
61 | group by sensorid
62 | order by Y DESC
63 | limit 10;
64 |
```

```
Query ef4f7849b5dc04b4:c5d0de2b00000000: 0% Complete (0 out of 8)
Query ef4f7849b5dc04b4:c5d0de2b00000000 100% Complete (8 out of 8)
Query ef4f7849b5dc04b4:c5d0de2b00000000 100% Complete (8 out of 8)
```

ef4f7849b5dc04b4:c5d0de2b00000000

Query History

Saved Queries

Results (10)

y	sensorid
1	2794
2	3724
3	817
4	2979
5	1169
6	4059
7	7628
8	3396
9	9360
10	2569

In my case the sensor with ID 2794 has 12 reading. Let's find out the min, max and average values for this sensor:

```
select min(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

```
68 | select min(sensorvalue) from improve2_sensor_table where sensorid=2794;
69 |
```

```
Query 744a0f8717c36b08:31cefd5c00000000: 62% Complete (5 out of 8)
Query 744a0f8717c36b08:31cefd5c00000000 100% Complete (8 out of 8)
```

744a0f8717c36b08:31cefd5c00000000

Query History

Saved Queries

Results (1)

min(sensorvalue)

```
1 | 66
```

```
select max(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

```
70| select max(sensorvalue) from improve2_sensor_table where sensorid=2794;  
71
```

Query 7741b899716233d0:a39bcd8600000000 100% Complete (8 out of 8)

[7741b899716233d0:a39bcd8600000000](#)

Query History Saved Queries Results (1)

max(sensorvalue)

1	252

```
select avg(sensorvalue) from improve2_sensor_table where sensorid=2794;
```

```
/1  
72| select avg(sensorvalue) from improve2_sensor_table where sensorid=2794;  
73
```

Query f445b364c5df79d1:7177c2de00000000: 62% Complete (5 out of 8)  
Query f445b364c5df79d1:7177c2de00000000 100% Complete (8 out of 8)

[f445b364c5df79d1:7177c2de00000000](#)

Query History Saved Queries Results (1)

avg(sensorvalue)

1	167.3333333333334

This concludes the IoT/Time series data and real-time streaming lab.



## APPENDIX - NiFi solution

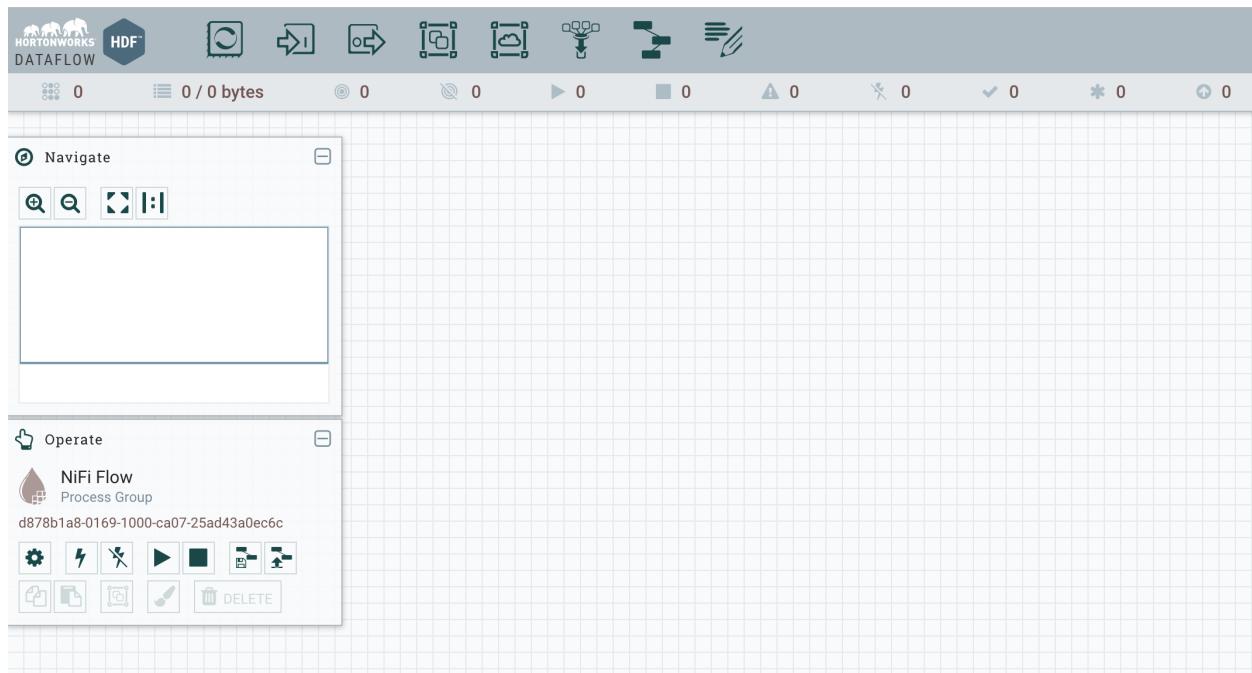
Here are the NiFi flows already implemented that can be imported.

Before to run them make sure you have changed the Kafka brokers and Kudu master urls to point to your cluster.

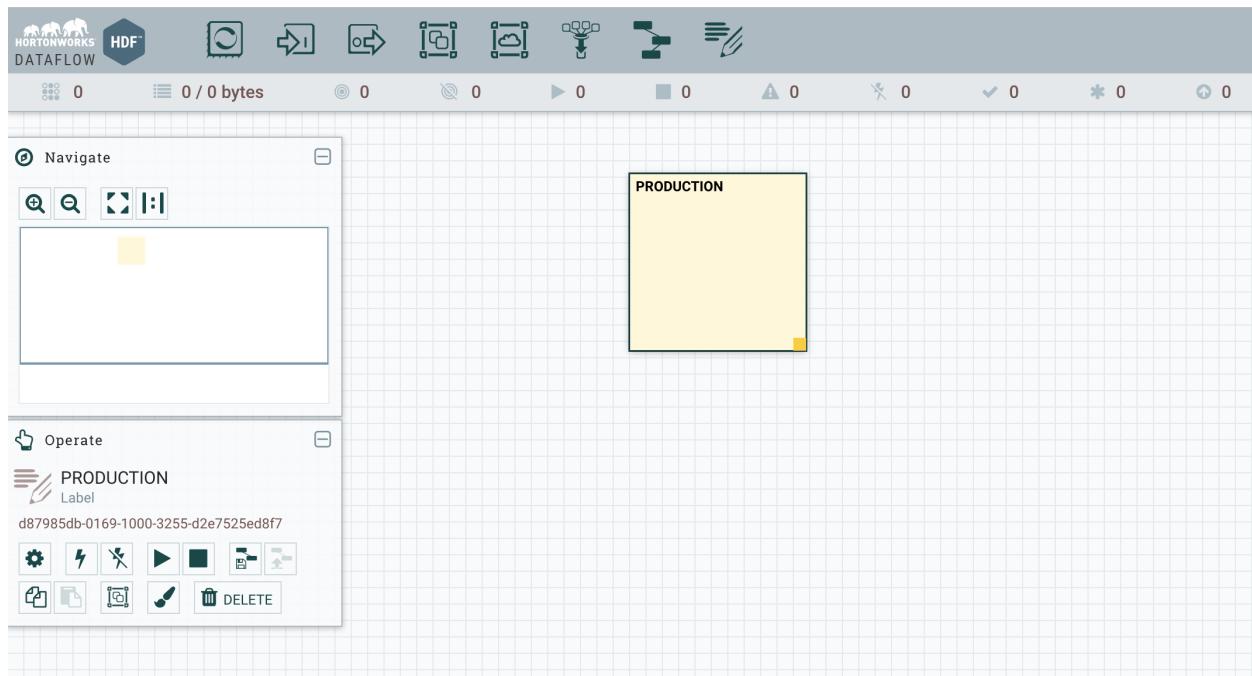
[generateDataSensors](#)  
[ingestDataSensors](#)

## APPENDIX - Promote a flow in Production

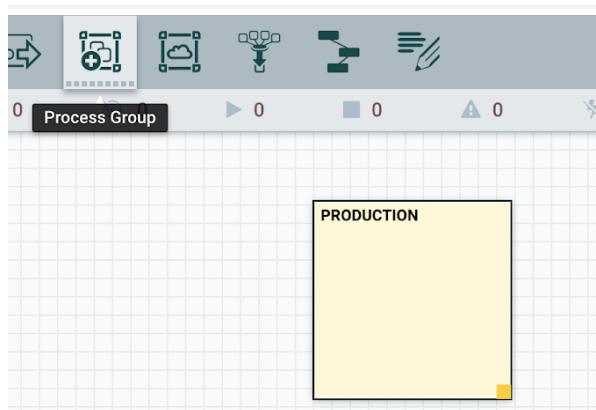
If you have a second NiFi installation to simulate the PRODUCTION environment, then connect to its Web UI:



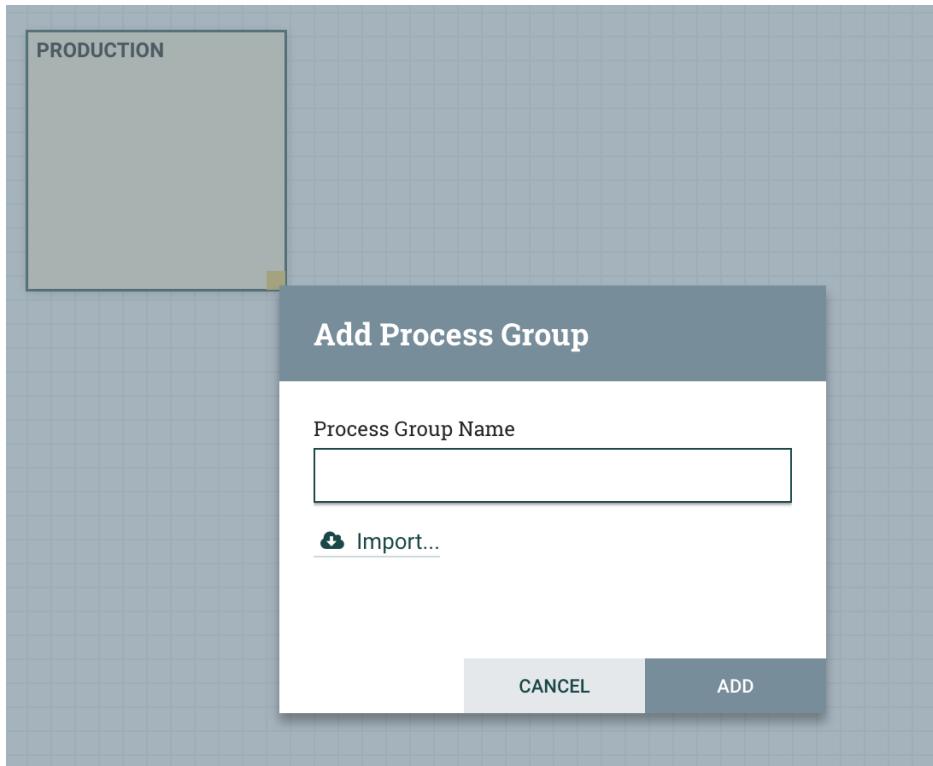
Add a label and called it PRODUCTION



Now connect the NiFi Registry, remember how we did before?  
Select the **Process Group** and ...



Drag&drop a Process Group to the canvas:



Select **Import**:

**Import Version**

Registry  
local

Bucket  
Demo

Name  
generateDataSensorsFlow

Version	Created	Comments
2	04/01/2019 10:27:20.880	changed the schedule from 10 sec to 5 sec
1	04/01/2019 10:25:32.134	initial version

CANCEL    IMPORT

Select the version 2

**Import Version**

Registry  
local

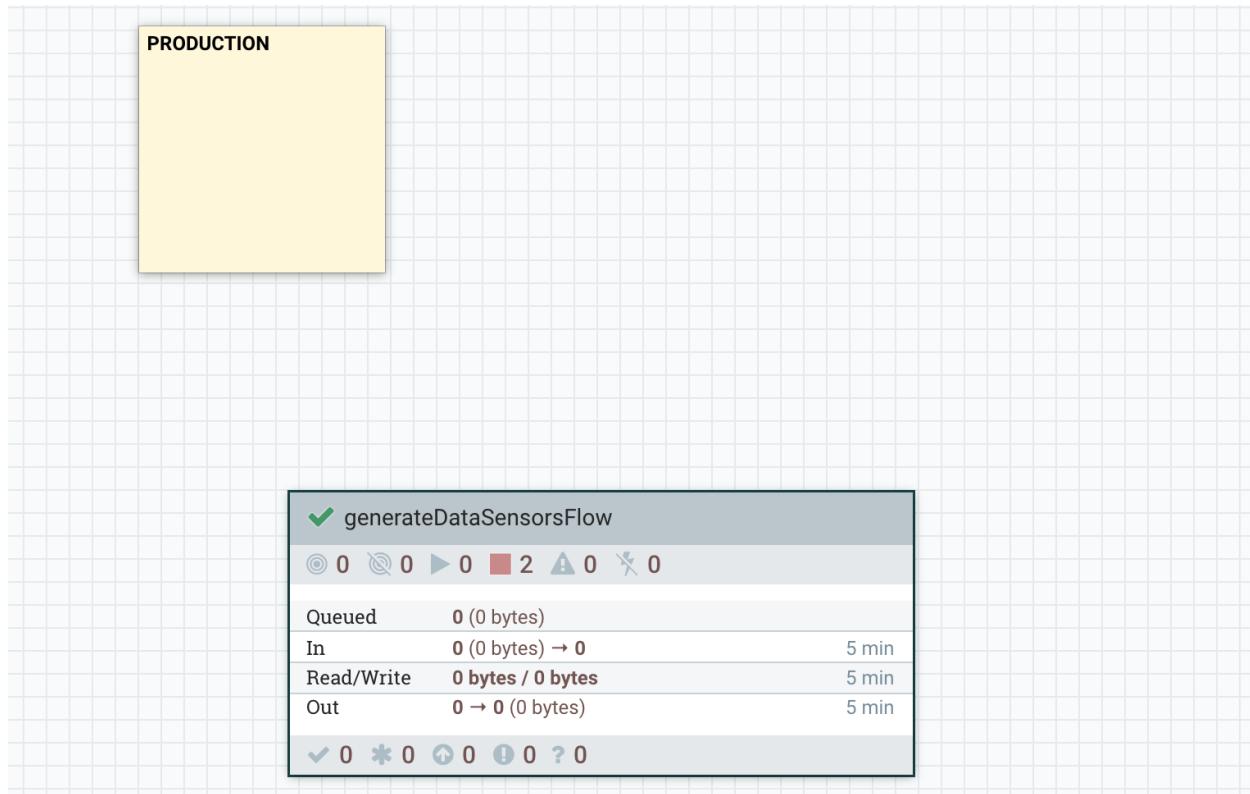
Bucket  
Demo

Name  
generateDataSensorsFlow

Version	Created	Comments
2	04/01/2019 10:27:20.880	changed the schedule from 10 sec to 5 sec
1	04/01/2019 10:25:32.134	initial version

CANCEL      IMPORT

And then **IMPORT**



we have an identical flow in a second NiFi instance.

---

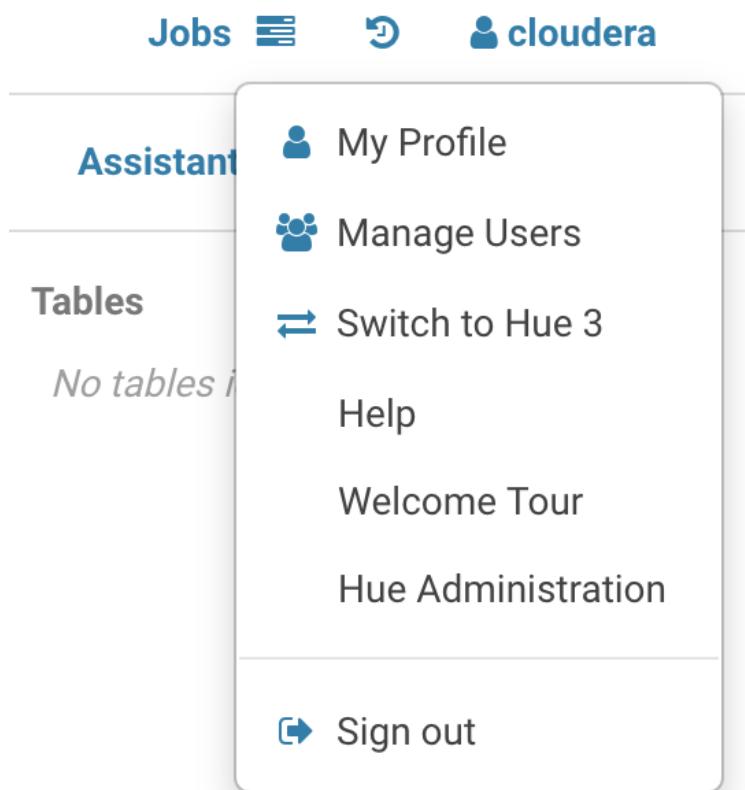
## APPENDIX - Using NiFi to ingest Web logs into EDH

### Getting the Web Logs file

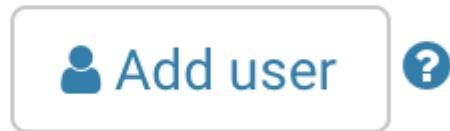
For this lab, for simplicity, we are going to upload a web logs file into HDFS and then we will configure a NiFi flow for splitting each line, reading it, extract the fields and then save them in Kudu. For real scenarios typically the external application will send each line as it is being generated to NiFi.

We are going to use the “*nifi*” user, to store and read the web logs file in HDFS, so let’s create it with the HUE Web UI.

Go to **Manage Users**:



Then Add User



with:

Username: *nifi*

Password: *nifi*

User Admin    Users    Groups    Permissions

## Hue Users - Create user

Step 1: Credentials (required)    Step 2: Profile and Groups    Step 3: Advanced

Username     New Password     Password confirmation

Create home directory

Back    Next    Add user

and then **Add user**.

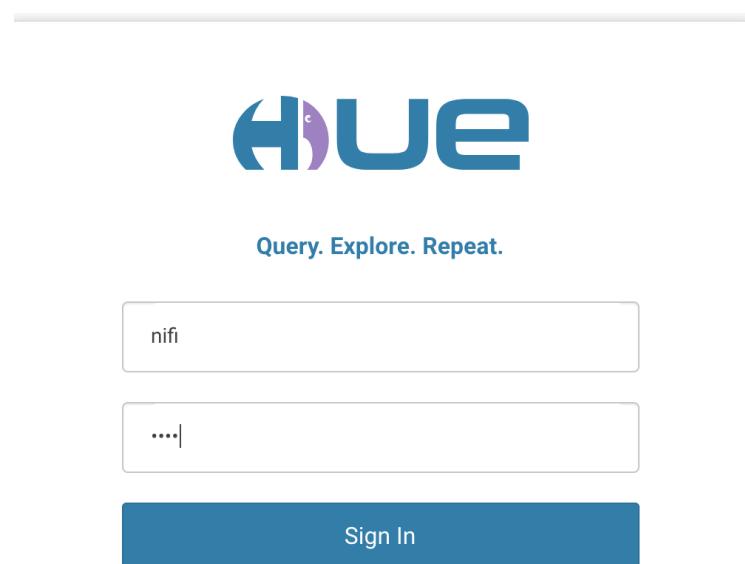
## Hue Users

Search for name, group, etc...		 Delete
<input type="checkbox"/>	Username	First Name
<input type="checkbox"/>	cloudera	
<input type="checkbox"/>	nifi	

Download the following file to your laptop **web.log.2**

<https://cloudera.box.com/s/52m5007eqx2asmx0l9c5tf9woazu64fh>

Then login to HUE with the *nifi* user



and upload the web logs file into the HDFS folder /user/nifi using the HUE File Browser.

The screenshot shows the Cloudera HUE interface. On the left, a sidebar menu lists 'Apps' (Editor, Scheduler), 'Browsers' (Documents), 'Files' (Tables, Indexes, Jobs). The 'Files' option is selected and highlighted in blue. In the main area, a modal window titled 'Upload to /user/nifi' is open, showing a progress bar for a file named 'web.log (1).2' at 3% from 37.8MB. Below the modal is a file listing table:

Name	Size	User	Group	Permissions	Date
..		hdfs	supergroup	drwxr-xr-x	May 03, 2019 01:18 AM
..		nifi	nifi	drwxr-xr-x	May 03, 2019 01:39 AM
web.log (1).2	37.8 MB	nifi	nifi	-rw-r--r-	May 03, 2019 01:39 AM

At the bottom of the table, it says 'Show 45 of 1 items' and 'Page 1 of 1'.

## Creating a Web Logs table

Open the Web log file to see how its lines look like:

```
19.133.215.123 - - [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
```

For each line we are going to extract the IP, the date, the URL, the HTTP version and so on ...and store these values inside a store, that for simplicity we have decided to be kudu. So in order to do that we need to create a kudu table for this purpose.

Within the Impala Query Editor type the following SQL statement:

```
CREATE TABLE weblogs_table
(
    ip STRING,
    log_date STRING,
    log_method STRING,
    url STRING,
    http_version STRING,
    code1 STRING,
    code2 STRING,
    user_agent STRING,
    PRIMARY KEY (ip,log_date)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

The screenshot shows the Impala Query Editor interface. At the top, there are tabs for 'Impala' and 'Add a name...', and buttons for 'Add a description...', 'Graph', 'Table', and 'More'. The status bar indicates '12.86s default text'.

```

1 CREATE TABLE weblogs_table
2 (
3   ip STRING,
4   log_date STRING,
5   log_method STRING,
6   url STRING,
7   http_version STRING,
8   code1 STRING,
9   code2 STRING,
10  user_agent STRING,
11  PRIMARY KEY (ip,log_date)
12 )
13 PARTITION BY HASH PARTITIONS 16
14 STORED AS KUDU;
15

```

Below the code, a message box says 'Success.' with a checkmark icon. At the bottom, there are links for 'Query History' and 'Saved Queries'.

A message box at the bottom displays the query text:

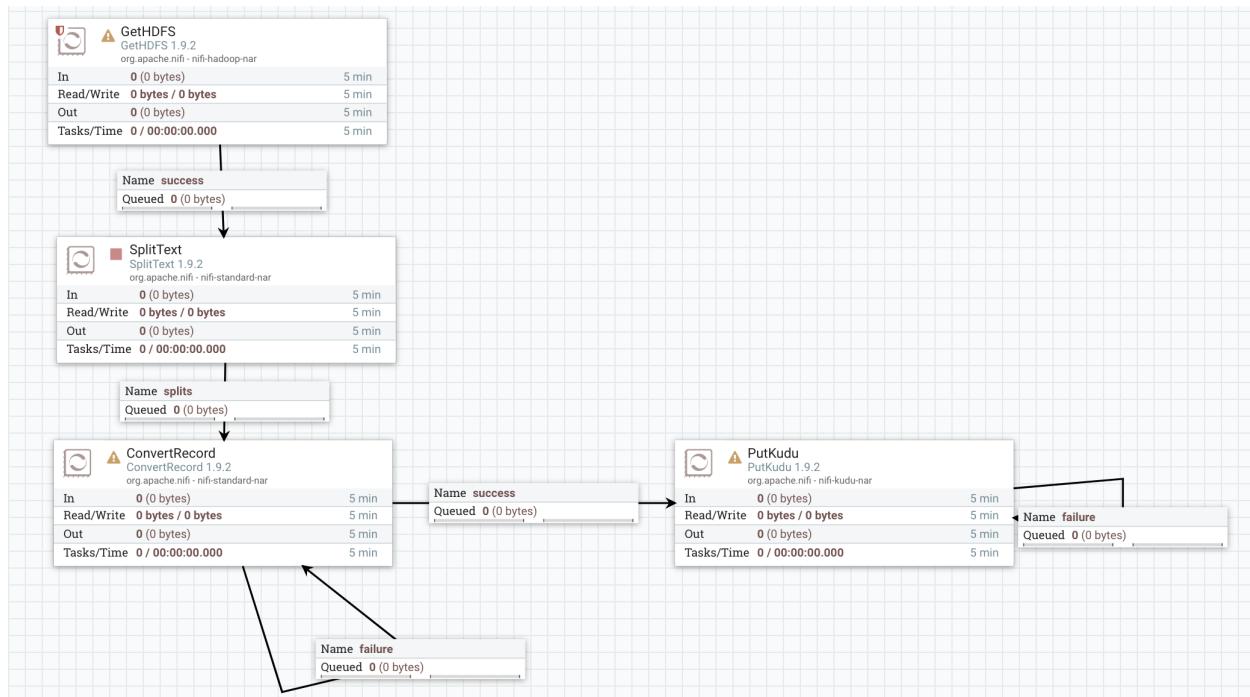
```

alcuni secondi fa ✓ CREATE TABLE weblogs_table ( ip STRING, log_date STRING, log_method STRING, url STRING, http_version STRING, code1 STRING, code2 STRING, user_agent STRING, PRIMARY KEY (ip,log_date) ) PARTITION BY HASH PARTITIONS 16 STORED AS KUDU

```

## Creating the NiFi Flow

Now we are ready to configure the following NiFi flow:

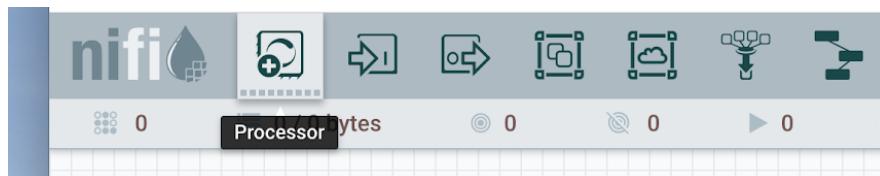


The flow should be able to do the following:

1. read the file from HDFS
2. split each line
3. extract the fields we are interested in
4. store them into kudu

Let's build it!

On the top-left corner, click on the **Processor** icon and drag it into the canvas:



and select **GetHDFS**:



Double click the processor and go to the **PROPERTIES** tab:

### Configure Processor

SETTINGS    SCHEDULING    PROPERTIES    COMMENTS

Required field

Property	Value
Hadoop Configuration Resources	/etc/hadoop/conf.cloudera.hdfs/hdfs-site.xml,/etc/hadoop/...
Kerberos Credentials Service	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
Kerberos Relogin Period	4 hours
Additional Classpath Resources	No value set
Directory	/user/nifi
Recurse Subdirectories	true
Keep Source File	false
File Filter Regex	No value set
Filter Match Name Only	true
Ignore Dotted Files	true
Minimum File Age	0 sec
Maximum File Age	No value set

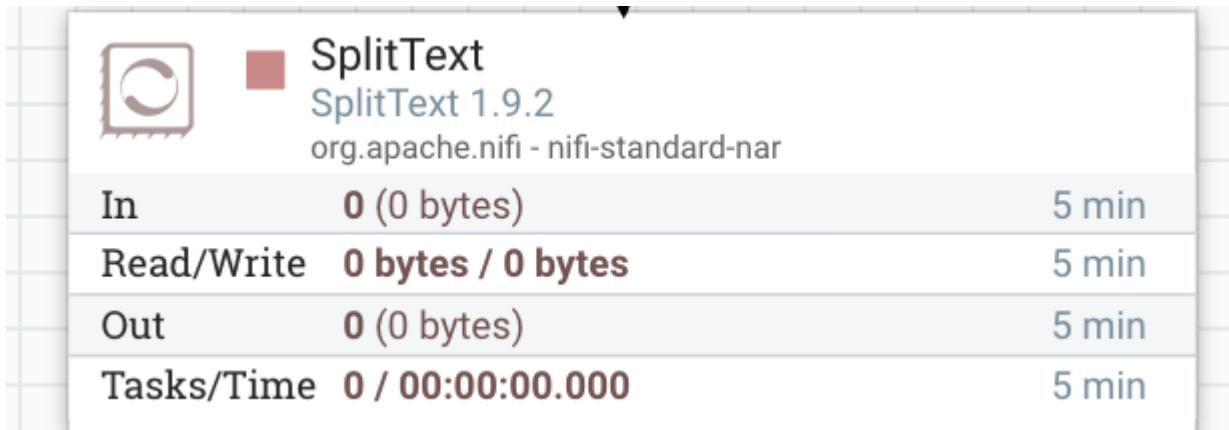
CANCEL    APPLY

Set the following properties values:

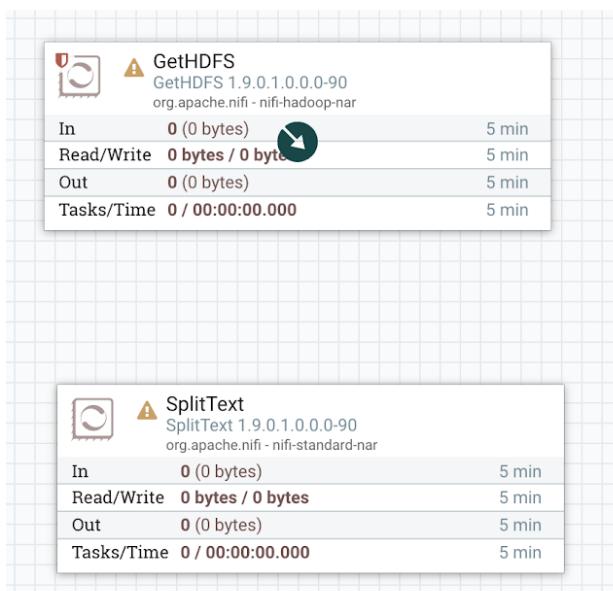
**Hadoop Configuration Resources:** /etc/hadoop/conf.cloudera.hdfs/hdfs-site.xml,/etc/hadoop/conf.cloudera.hdfs/core-site.xml

**Directory:** /user/nifi

Then add another processor to the flow, the **SplitText**:



and connect the two processor. With the mouse, go over the **GetHDFS** step and you should see an arrow icon and drag it to the **SplitText**:

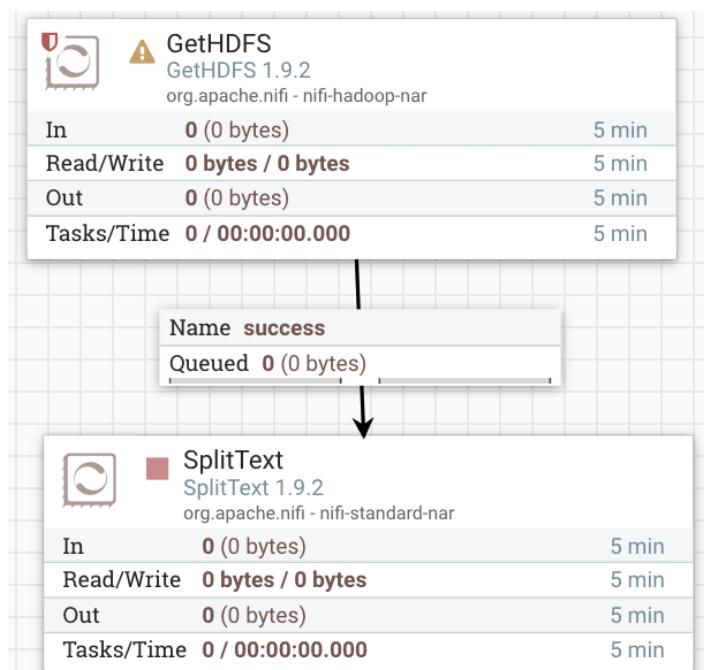


accept the default

### Create Connection

<b>DETAILS</b>	<b>SETTINGS</b>
From Processor <b>GetHDFS</b> GetHDFS  Within Group NiFi Flow  For Relationships <input checked="" type="checkbox"/> success	To Processor <b>SplitText</b> SplitText  Within Group NiFi Flow
<a href="#">CANCEL</a> <a href="#">ADD</a>	

and click ADD



Double click on the **SplitText** and go to the **PROPERTIES** tab:

The screenshot shows the 'Configure Processor' dialog with the 'PROPERTIES' tab selected. A table lists properties and their values:

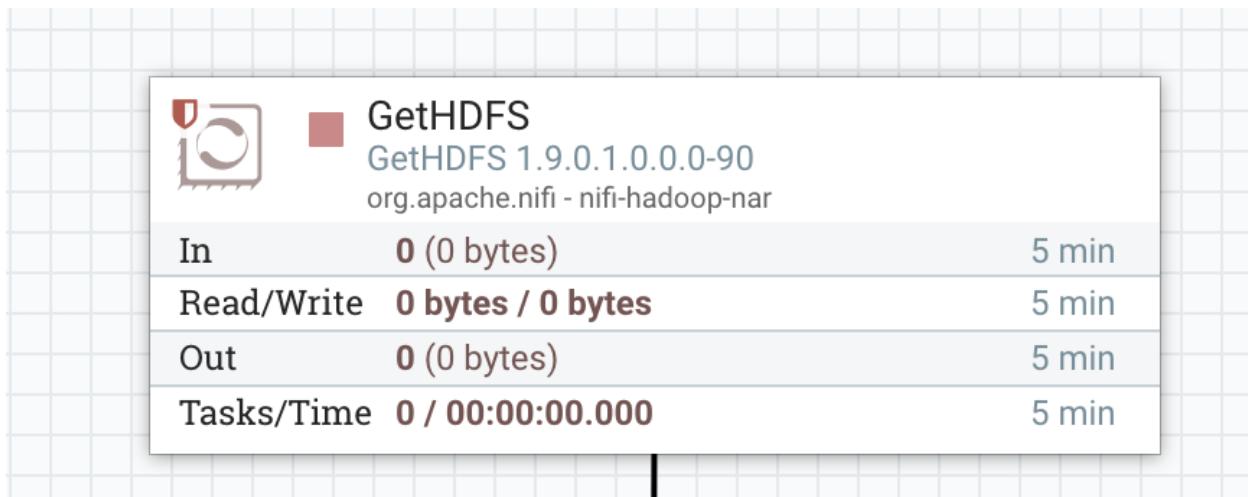
Property	Value
Line Split Count	1
Maximum Fragment Size	No value set
Header Line Count	0
Header Line Marker Characters	No value set
Remove Trailing Newlines	true

At the bottom right are 'CANCEL' and 'APPLY' buttons.

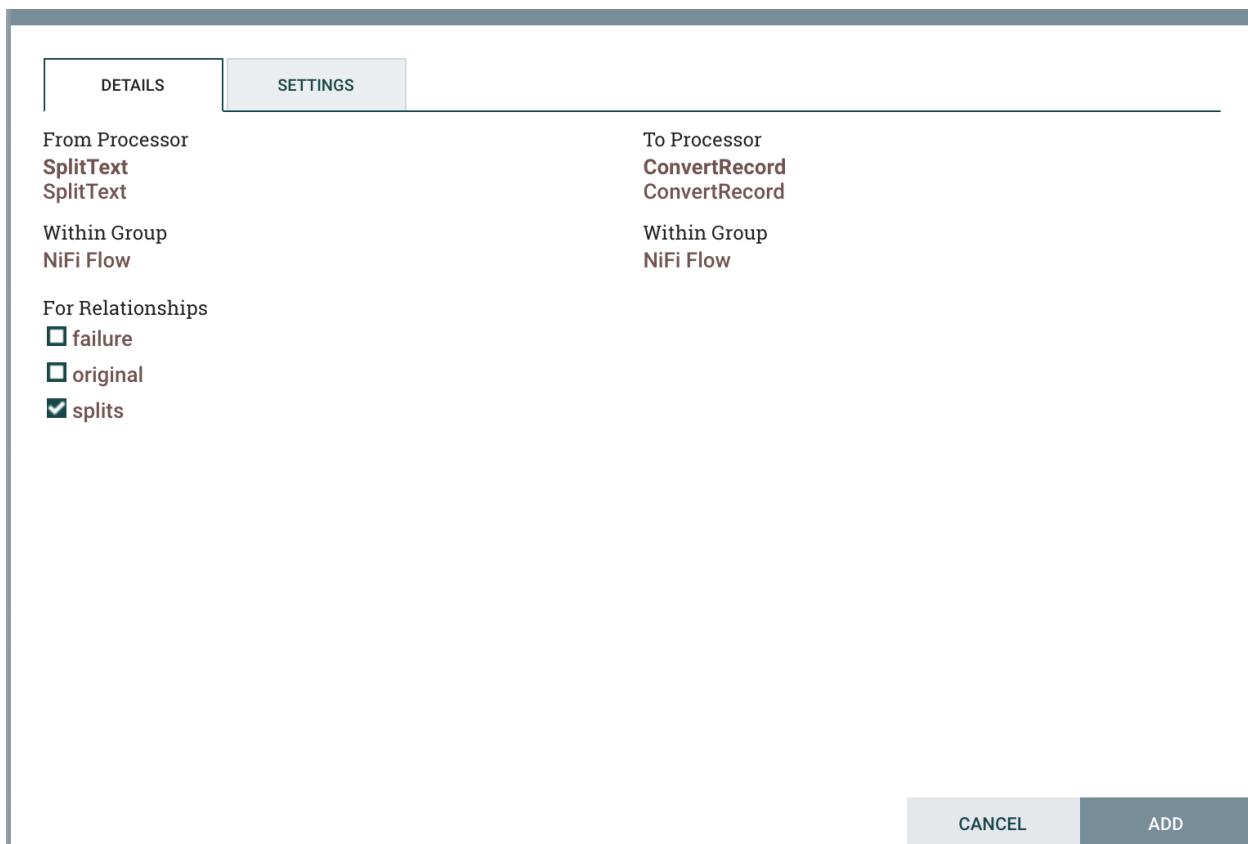
Set 1 to the **Line Split Count** and click **APPLY**.

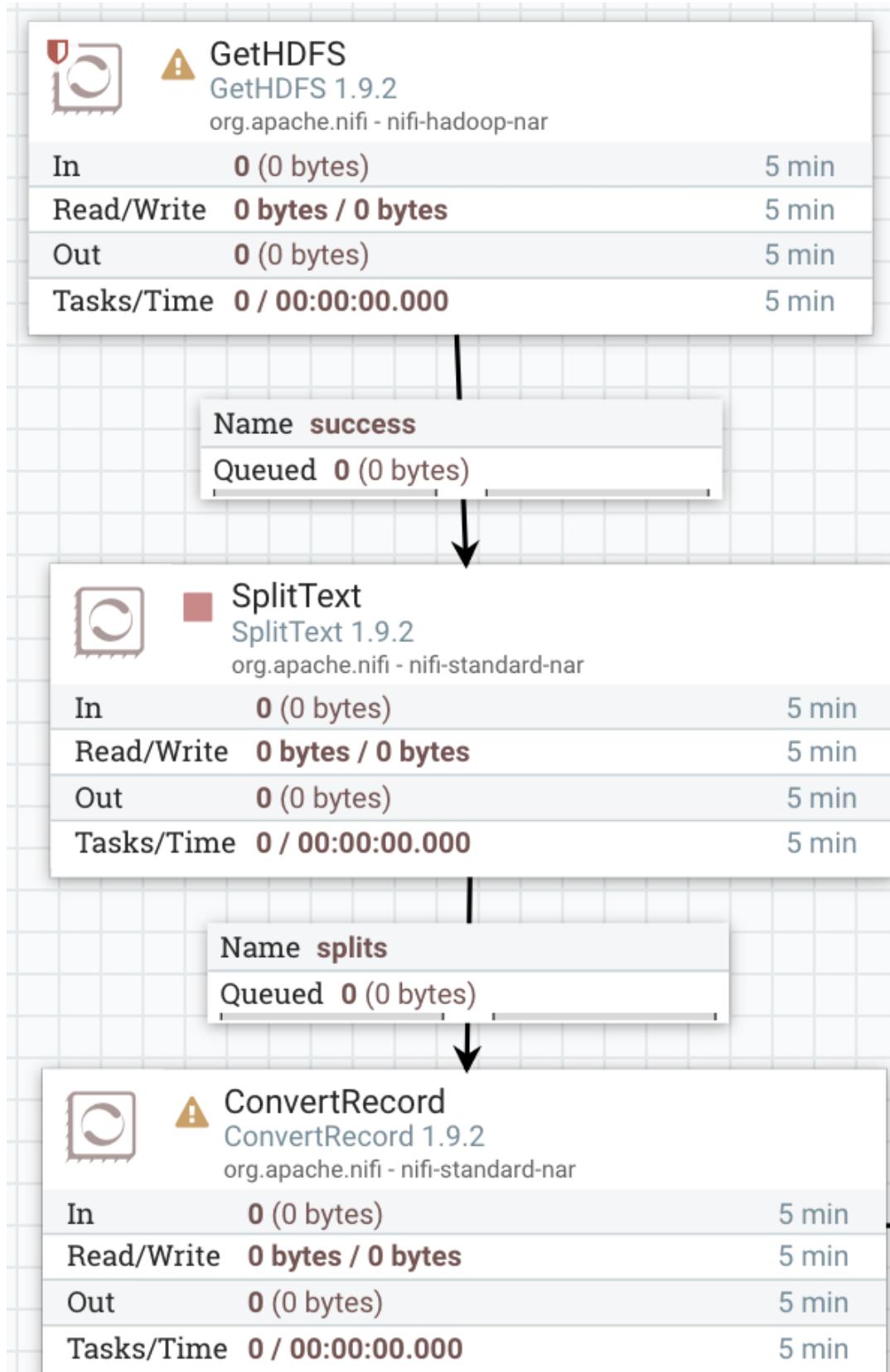
This step will extract one line at the time and pass it forward to the next step, that is going to understand and detect the fields (IP, URL, ...).

After this step the icon on the left top corner of GetHDFS should be a square indicating that was able to access and get the hdfs directory:



Add a **ConvertRecord** processor and connect to the **SplitText**, by setting the **splits** option in the connection details:

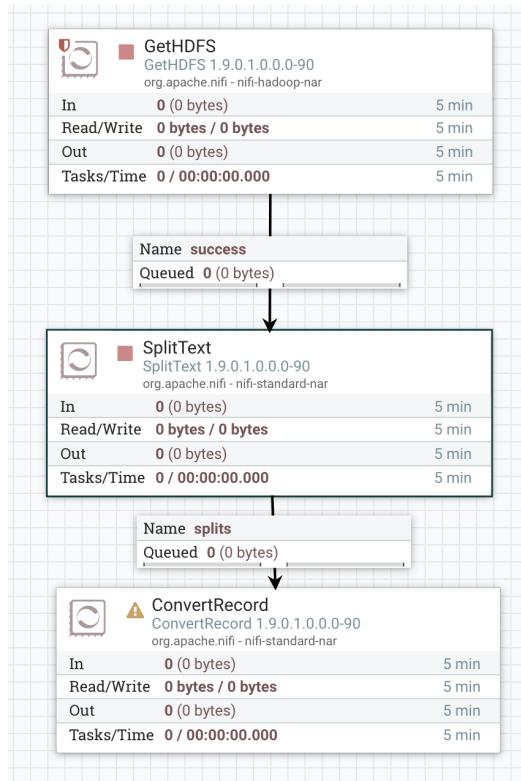




Double click on the **SplitText**, **SETTINGS** tab and check **failure** and **original** for **Automatically Terminate Relationships**

The screenshot shows the 'Configure Processor' dialog for a 'SplitText' processor. The 'SETTINGS' tab is active. On the left, there are fields for Name (SplitText), Id (49784ebc-54a7-3834-b4fb-1228fa4021cc), Type (SplitText 1.9.2), Bundle (org.apache.nifi - nifi-standard-nar), Penalty Duration (30 sec), Yield Duration (1 sec), and Bulletin Level (WARN). On the right, under 'Automatically Terminate Relationships', two checkboxes are checked: 'failure' (described as routing the original file to this destination if splitting fails) and 'original' (described as routing the original input file to this destination when it has been successfully split). There is also an unchecked checkbox for 'splits'. At the bottom right are 'CANCEL' and 'APPLY' buttons.

And now also the **SplitText** has a red square:



Double click on the **ConvertRecord** and go to the **PROPERTIES** tab:

**Configure Processor**

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS											
<p><b>Required field</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">Property</th> <th style="width: 50%;">Value</th> </tr> </thead> <tbody> <tr> <td>Record Reader</td> <td>?</td> <td>No value set</td> </tr> <tr> <td>Record Writer</td> <td>?</td> <td>No value set</td> </tr> <tr> <td>Include Zero Record FlowFiles</td> <td>?</td> <td>true</td> </tr> </tbody> </table>				Property	Value	Record Reader	?	No value set	Record Writer	?	No value set	Include Zero Record FlowFiles	?	true
Property	Value													
Record Reader	?	No value set												
Record Writer	?	No value set												
Include Zero Record FlowFiles	?	true												

The **Record Reader** property specify how to read the incoming data, while the **Record Write** how to write out the data.

Because the logs line look like the following:

```
79.133.215.123 - - [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
```

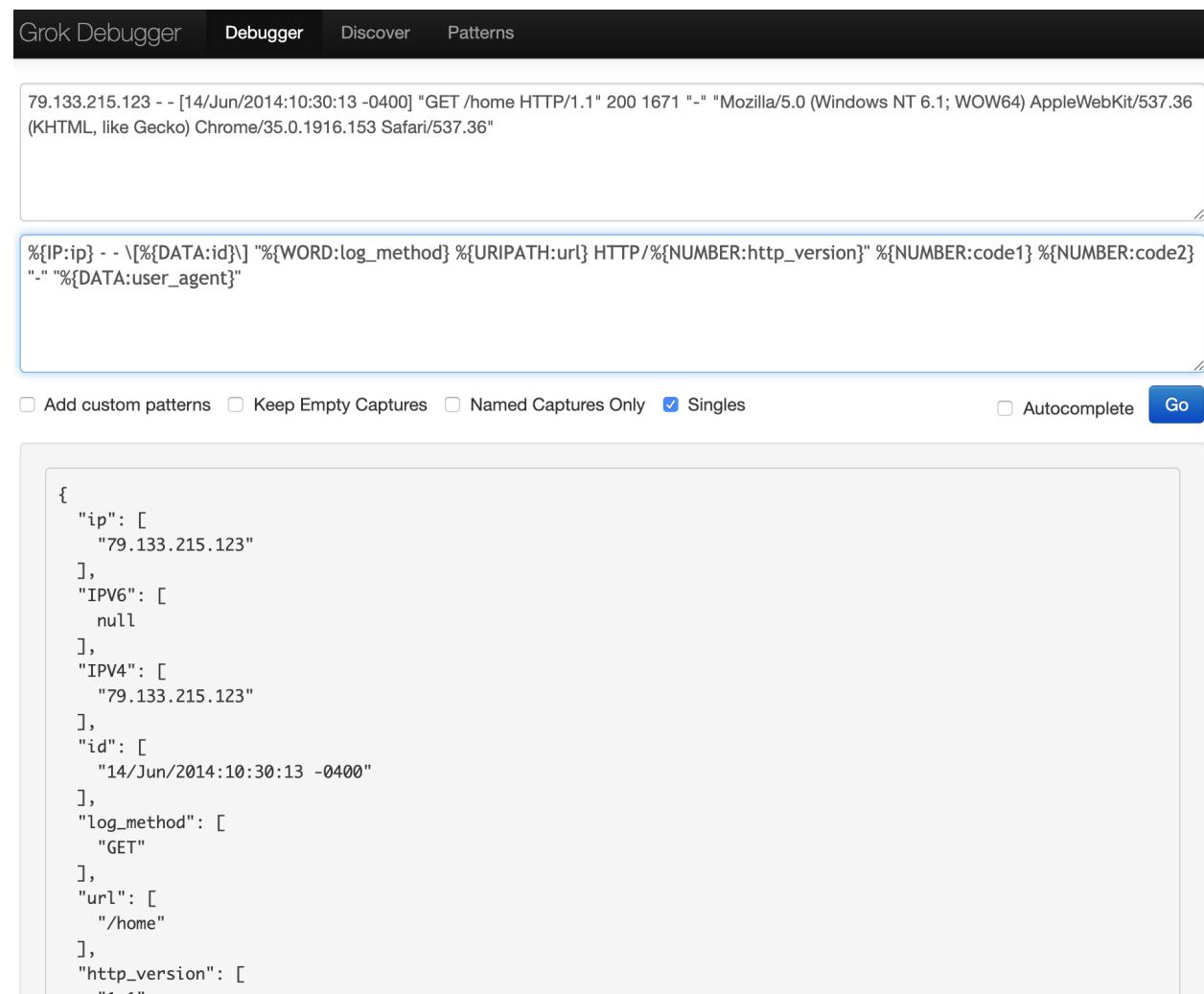
we are going to use GROK.

In order to correctly write the GROK expression, go to this GROK debugger  
<https://grokdebug.herokuapp.com/>

Paste the logs line and then this GROK expression:

```
%{IP:ip} -- \[%{DATA:id}\] "%{WORD:log_method} %{URIPATH:url}  
HTTP/%{NUMBER:http_version}" %{NUMBER:code1} %{NUMBER:code2} "-"  
"%{DATA:user_agent}"
```

and you should see our fields extracted:

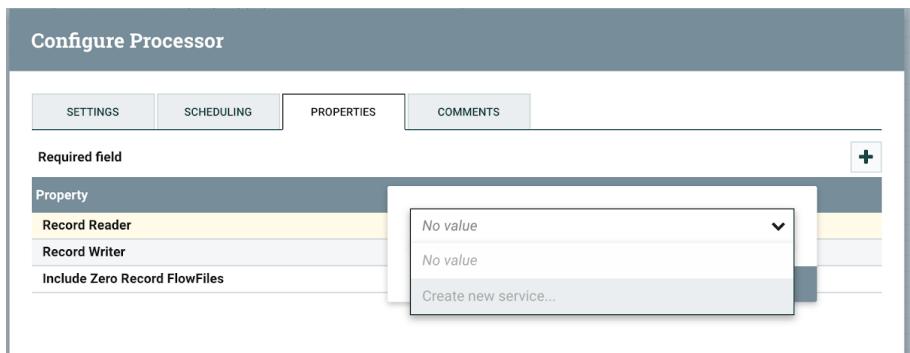


The screenshot shows the Grok Debugger interface. At the top, there are tabs: 'Grok Debugger' (selected), 'Debugger', 'Discover', and 'Patterns'. Below the tabs, a log line is displayed: "79.133.215.123 -- [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"". Underneath the log line, the GROK pattern is shown: "%{IP:ip} -- \[%{DATA:id}\] "%{WORD:log\_method} %{URIPATH:url} HTTP/%{NUMBER:http\_version}" %{NUMBER:code1} %{NUMBER:code2} "-" "%{DATA:user\_agent}"". Below the pattern, the extracted fields are listed in a JSON-like structure:

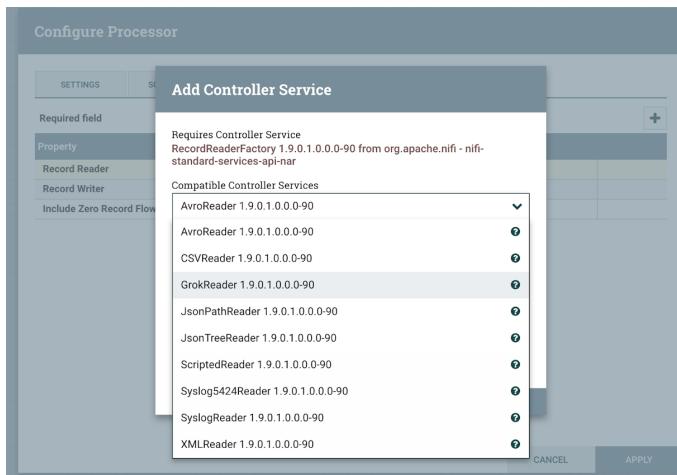
```
{  
  "ip": [  
    "79.133.215.123"  
  ],  
  "IPV6": [  
    null  
  ],  
  "IPV4": [  
    "79.133.215.123"  
  ],  
  "id": [  
    "14/Jun/2014:10:30:13 -0400"  
  ],  
  "log_method": [  
    "GET"  
  ],  
  "url": [  
    "/home"  
  ],  
  "http_version": [  
    "1 1"  
  ]  
}
```

At the bottom of the interface, there are several checkboxes: 'Add custom patterns', 'Keep Empty Captures', 'Named Captures Only', 'Singles' (which is checked), 'Autocomplete', and a 'Go' button.

Click on the **Value** of the **Record Reader** and select **Create new service...**



Select GrokReader and give it a name (for example GrokReaderWeblogs...)



### Add Controller Service

Requires Controller Service  
RecordReaderFactory 1.9.2 from org.apache.nifi - nifi-standard-services-api-nar

Compatible Controller Services

GrokReader 1.9.2

Controller Service Name

GrokReader

Bundle  
org.apache.nifi - nifi-record-serialization-services-nar

Tags  
logstash, regex, reader, grok, logfiles, unstructured, record, pattern, parse, text, logs

CANCEL CREATE

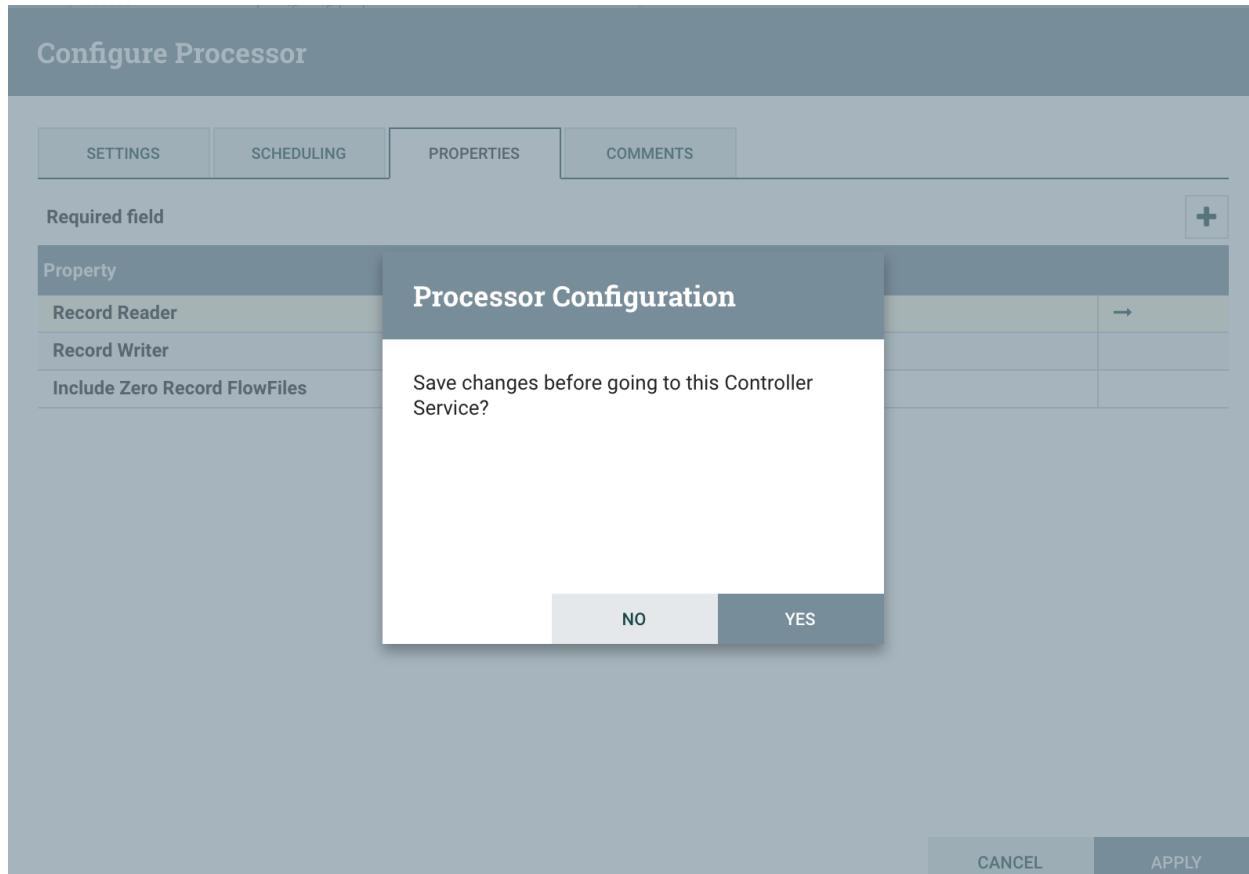
### Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value	
Record Reader	GrokReaderWeblogs	→
Record Writer	No value set	
Include Zero Record FlowFiles	true	

Then click on the arrow on the right, and save changes



and then click on the **gear icon**, for configuring it:

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Name: GrokReaderWeblogs

Referencing Components: !

Processors (1)

ConvertRecord ConvertRecord

Id: 7ce8653a-016a-1000-0000-00002c854f2d

Type: GrokReader 1.9.0.1.0.0.0-90

Bundle: org.apache.nifi - nifi-record-serialization-services-nar

Supports Controller Service

- RecordReaderFactory 1.9.0.1.0.0.0-90 from org.apache.nifi - nifi-standard-services-api-nar

CANCEL APPLY

Go to the **PROPERTIES** tab

The screenshot shows the 'Configure Controller Service' interface with the 'PROPERTIES' tab selected. A table lists properties and their values:

Property	Value
Schema Access Strategy	Use String Fields From Grok Expression
Schema Registry	No value set
Schema Name	\$(schema.name)
Schema Version	No value set
Schema Branch	No value set
Schema Text	\$(avro.schema)
Grok Pattern File	No value set
Grok Expression	No value set
No Match Behavior	Append to Previous Message

At the bottom right are 'CANCEL' and 'APPLY' buttons.

and set these properties:

### Schema Access Strategy: Use 'Schema Name' Property

**Schema Name:** weblogs

**Grok Expression:**

```
%{IP:ip} -- \[%{DATA:log_date}\] "%{WORD:log_method}
%{URIPATH:url} HTTP/%{NUMBER:http_version}" %{NUMBER:code1}
%{NUMBER:code2} "-" "%{DATA:user_agent}"
```

The screenshot shows the 'Configure Controller Service' interface with the 'PROPERTIES' tab selected. A table lists properties and their values, with 'Schema Name' explicitly set to 'weblogs':

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	No value set
Schema Name	weblogs
Schema Version	No value set
Schema Branch	No value set
Schema Text	\$(avro.schema)
Grok Pattern File	No value set
Grok Expression	%{IP:ip} -- \[%{DATA:log_date}\] "%{WORD:log_m...
No Match Behavior	Append to Previous Message

CANCEL      APPLY

Also for the **Schema Registry** select **Create new service...**

Configure Controller Service

SETTINGS    PROPERTIES    COMMENTS

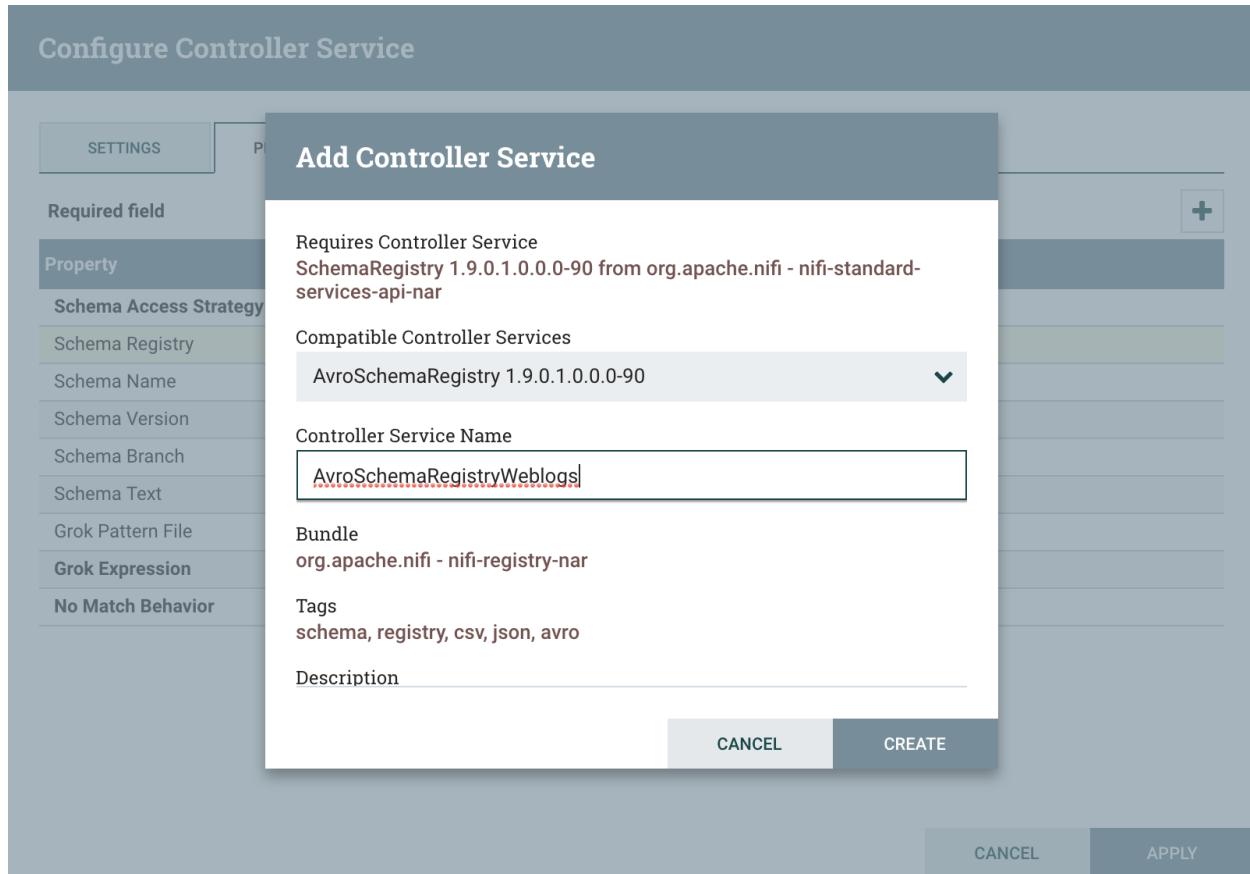
Required field

Property	Value
Schema Access Strategy	No value
Schema Registry	No value
Schema Name	
Schema Version	
Schema Branch	
Schema Text	\$(avro.schema)
Grok Pattern File	No value set
Grok Expression	%(IP:ip) - - \[%{DATA:log_date}\] "%{WORD:log_m...}
No Match Behavior	Append to Previous Message

+ Create new service...

CANCEL      APPLY

and select an **AvroSchemaRegistry**, give it a name (for example AvroSchemaRegistryWeblogs)



click **CREATE**

Configure Controller Service

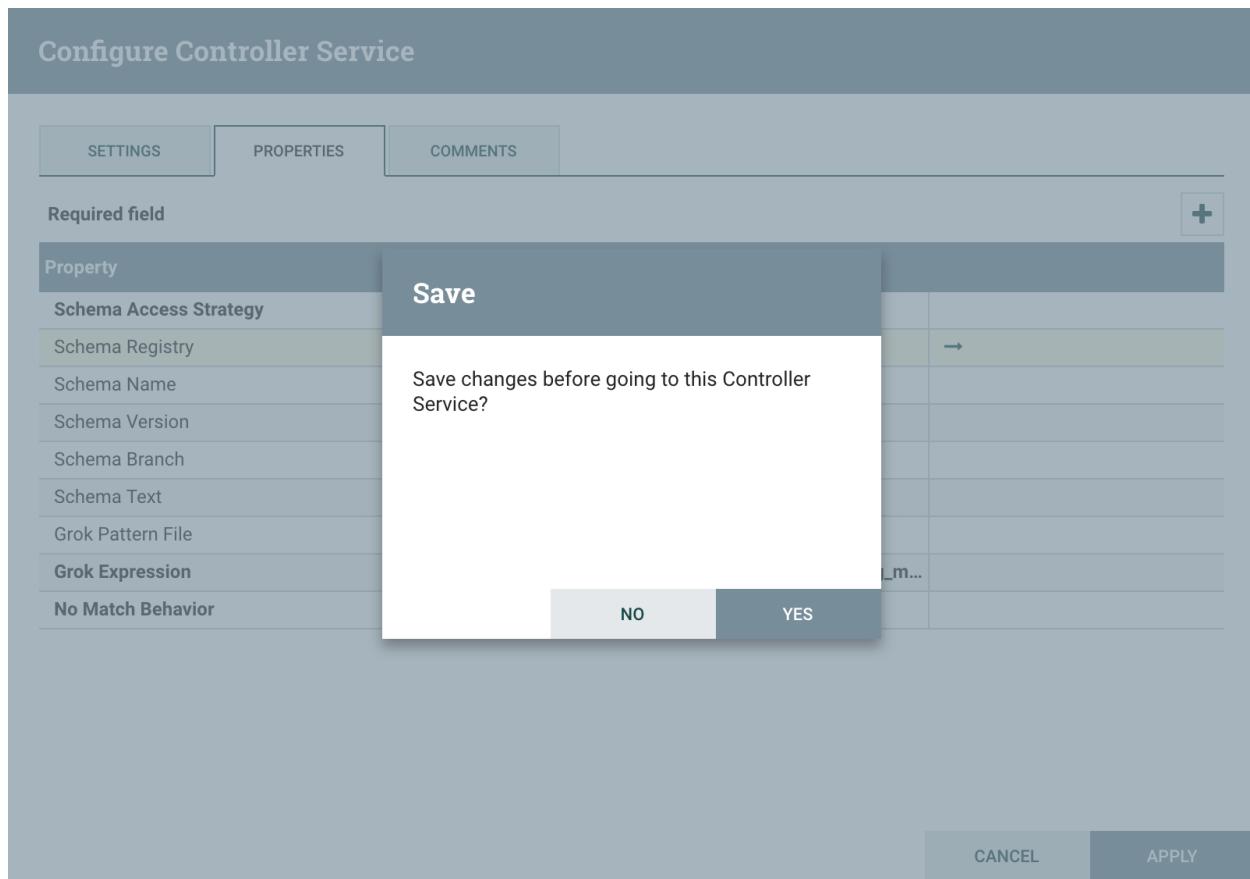
SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	AvroSchemaRegistryWeblogs →
Schema Name	weblogs
Schema Version	No value set
Schema Branch	No value set
Schema Text	\${avro.schema}
Grok Pattern File	No value set
Grok Expression	%{IP:ip} -- \[%{DATA:log_date}\] "%{WORD:log_m...}
No Match Behavior	Append to Previous Message

CANCEL APPLY

and configure as follows by clicking the arrow:



Save changes.

The screenshot shows the 'NiFi Flow Configuration' interface. It has tabs for 'GENERAL' and 'CONTROLLER SERVICES'. Under 'CONTROLLER SERVICES', there is a table listing two services:

Name	Type	Bundle	State	Scope
AvroSchemaRegistryWeblogs	AvroSchemaRegistry 1.9.0.0.0-90	org.apache.nifi - nifi-registry-nar	Disabled	NiFi Flow
GrokReaderWeblogs	GrokReader 1.9.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Invalid	NiFi Flow

Click the gear icon for configuring the **AvroSchemaRegistryWeblogs**:

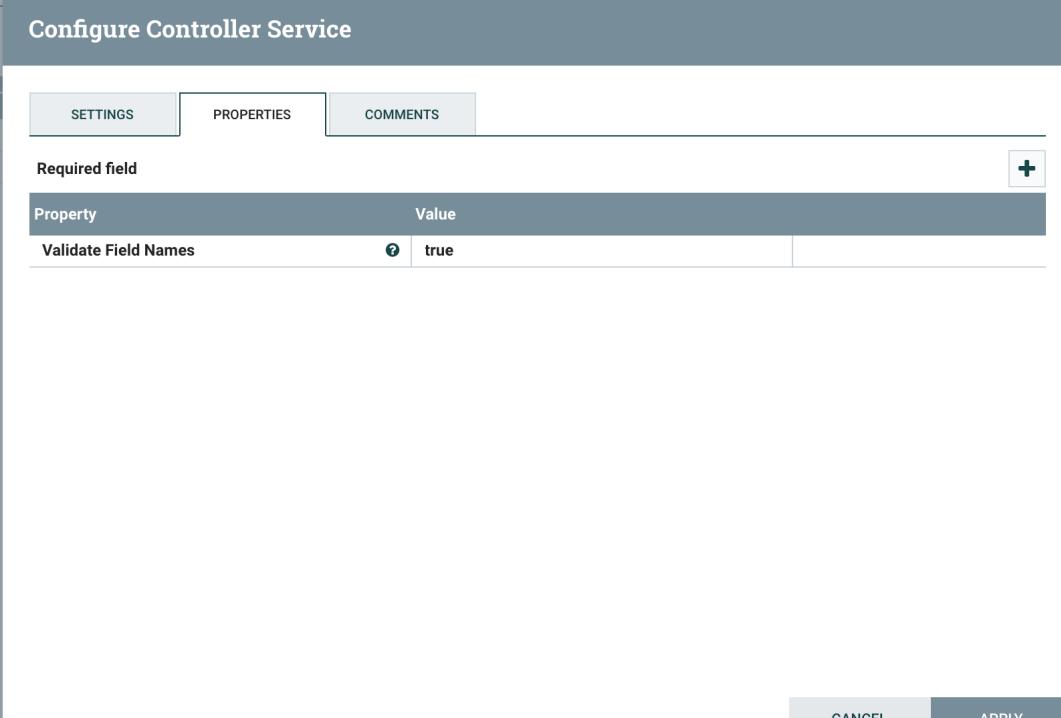
Configure Controller Service

SETTINGS PROPERTIES COMMENTS

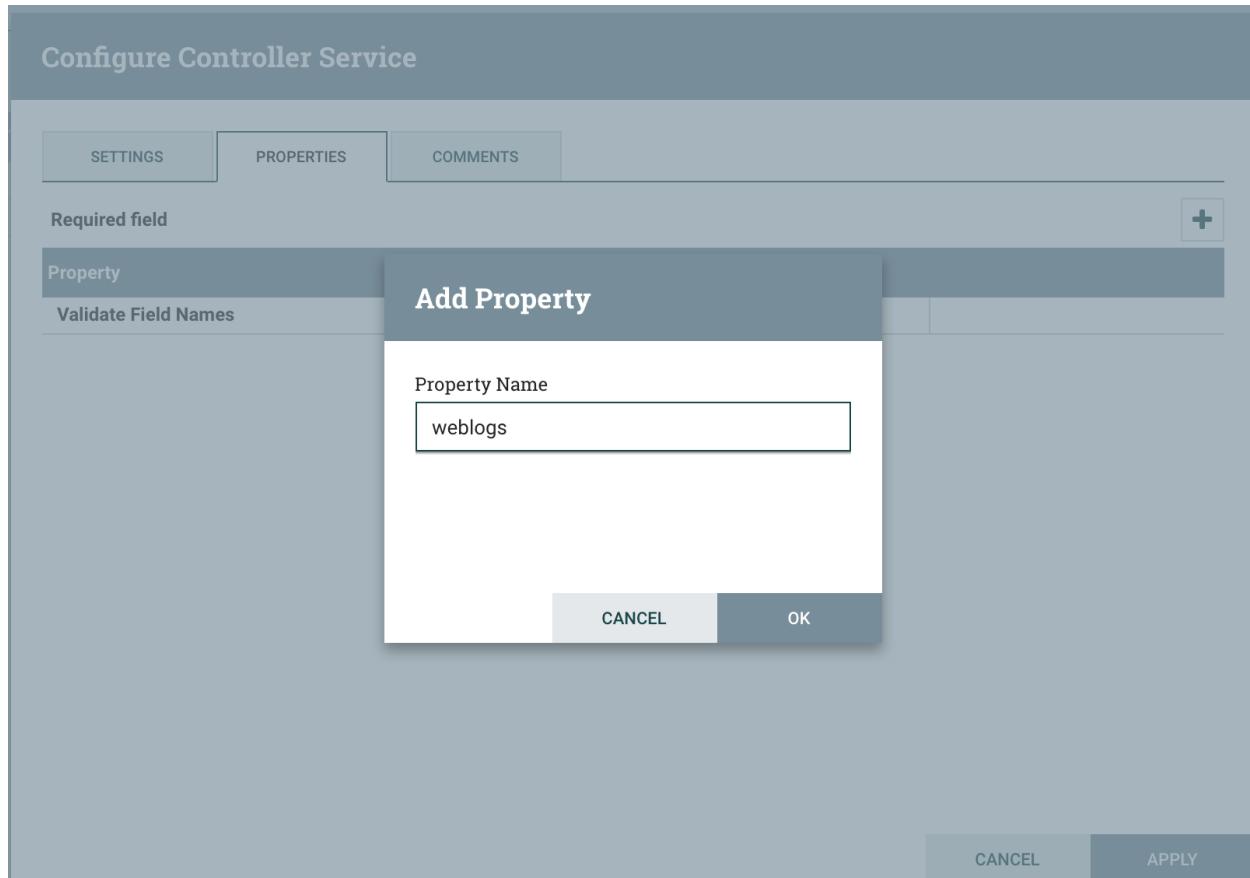
Required field +

Property	Value
Validate Field Names	true

CANCEL APPLY

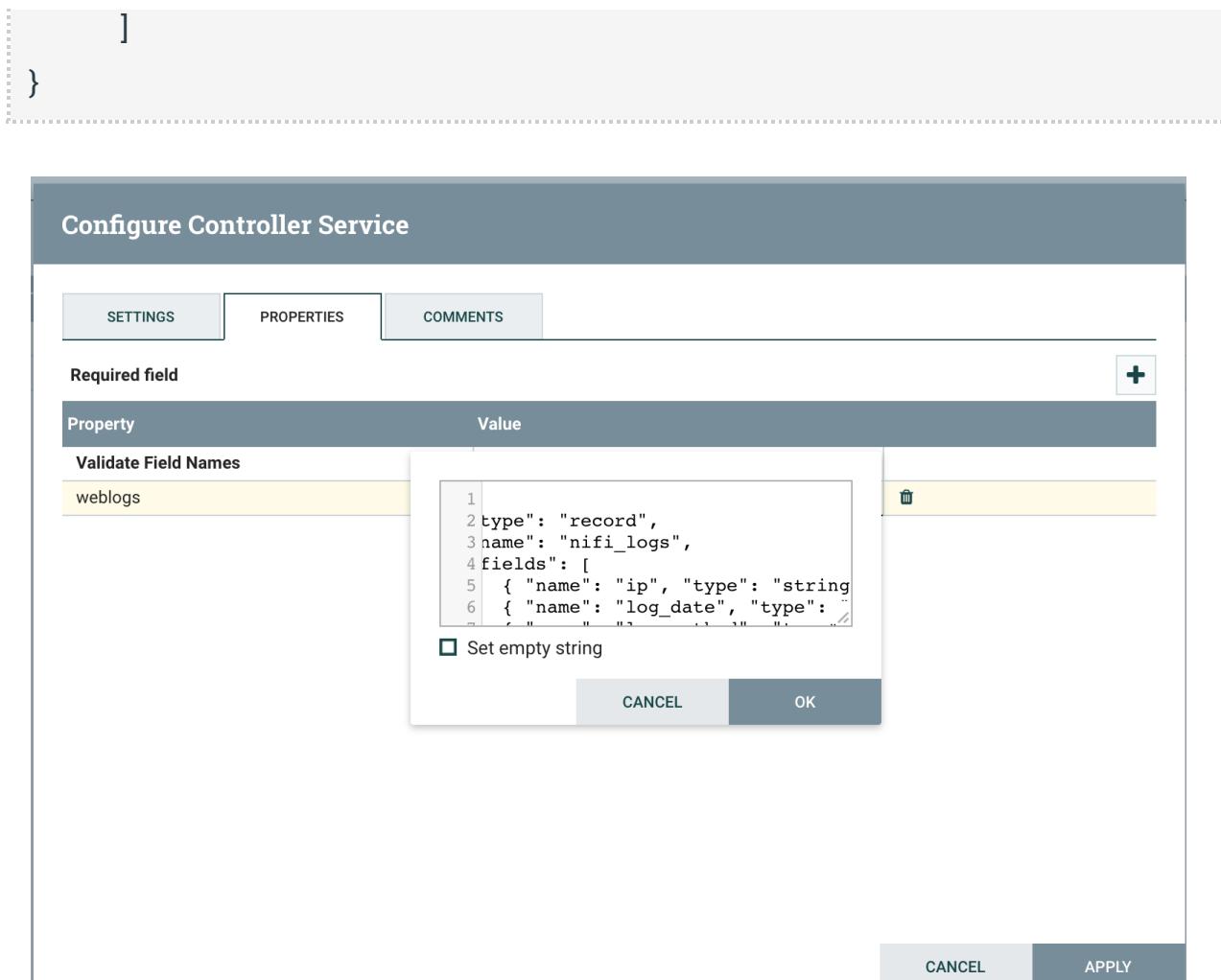


Click the **Plus (+) icon** to add a property *weblogs*



with value below that is the schema that describes our record.

```
{  
  "type": "record",  
  "name": "nifi_logs",  
  "fields": [  
  
    {"name": "ip", "type": "string"},  
    {"name": "log_date", "type": "string"},  
    {"name": "log_method", "type": "string"},  
    {"name": "url", "type": "string"},  
    {"name": "http_version", "type": "string"},  
    {"name": "code1", "type": "string"},  
    {"name": "code2", "type": "string"},  
    {"name": "user_agent", "type": "string"},  
  ]}
```



click OK

Configure Controller Service

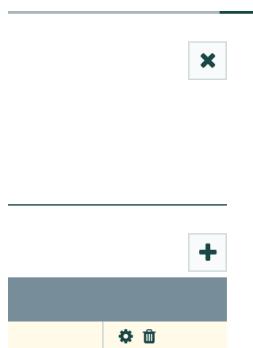
SETTINGS PROPERTIES COMMENTS

Required field +

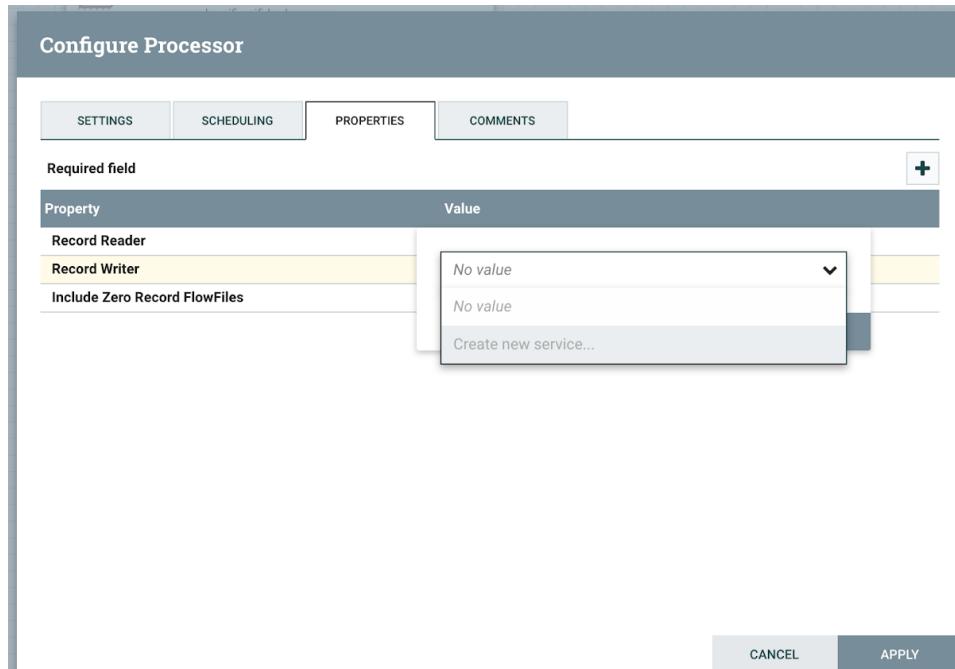
Property	Value
Validate Field Names	true
weblogs	{ "type": "record", "name": "nifi_logs", "fields": [ { "na... <span style="border: 1px solid #ccc; padding: 2px;">X</span>

CANCEL APPLY

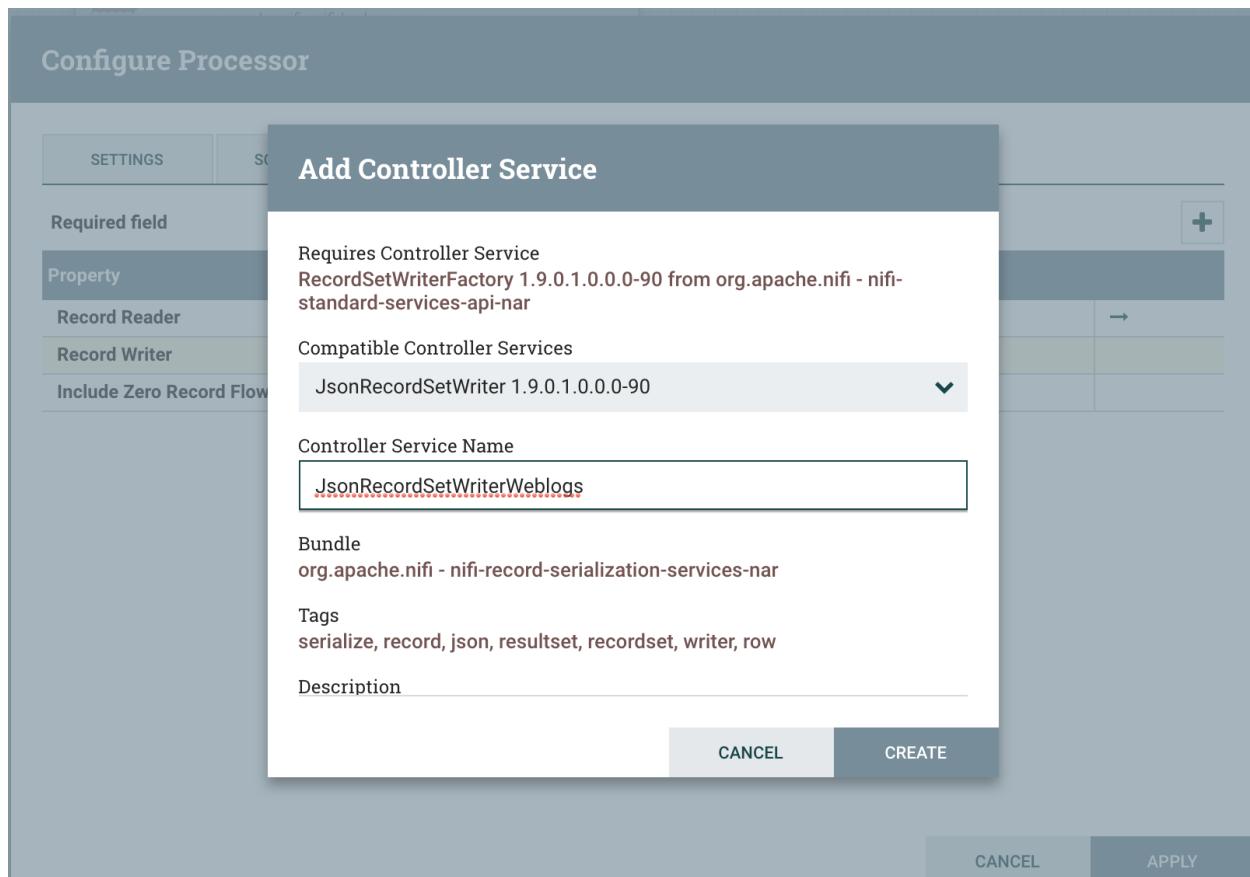
Then click **APPLY** and click on the **X** icon



Then go back to the **ConvertRecord** and for the **Record Writer** create a new service



of type **JsonRecordSetWriter** (for example `JsonRecordSetWriterWeblogs`):



click **CREATE**

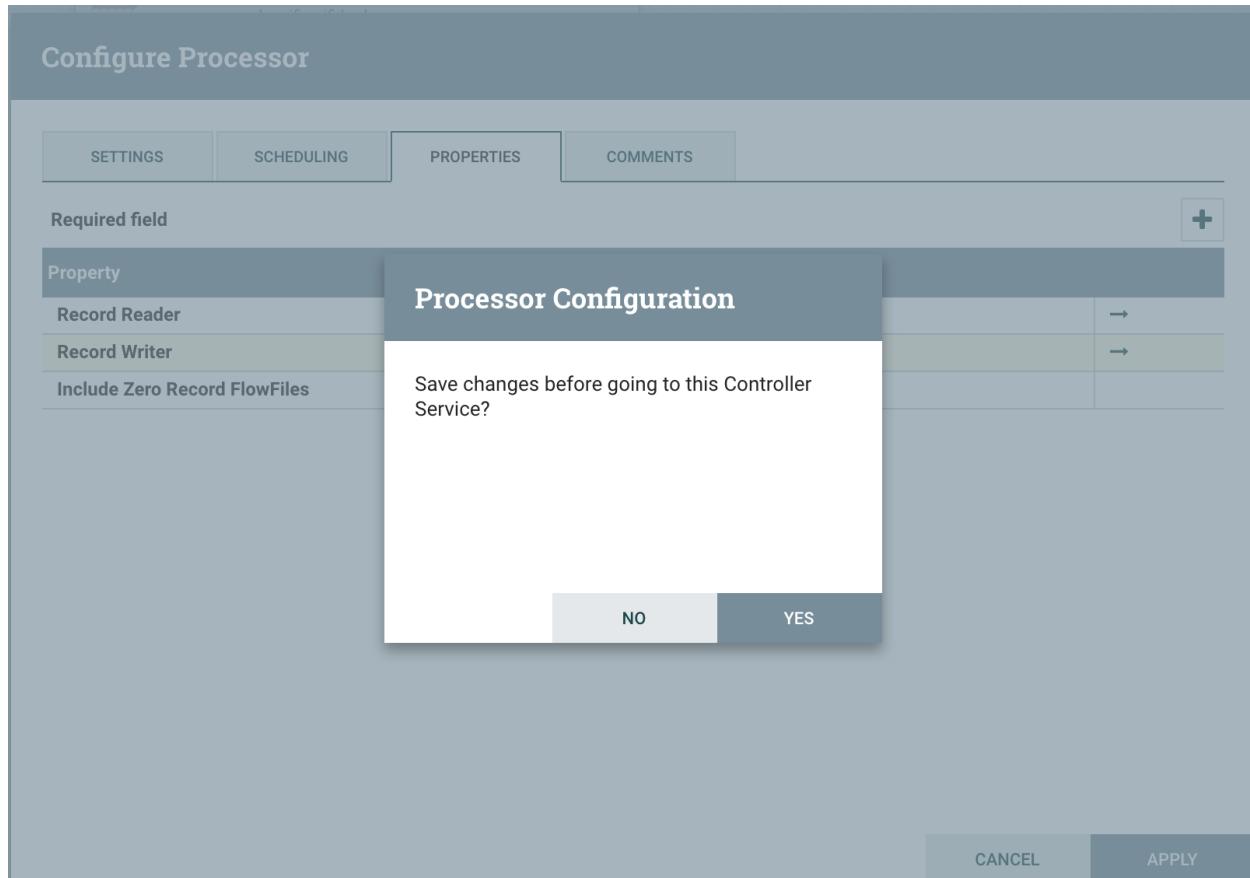
Property	Value
Record Reader	GrokReaderWeblogs
Record Writer	JsonRecordSetWriterWeblogs
Include Zero Record FlowFiles	true

Required field

+

CANCEL      APPLY

then configure by clicking the arrow and remember to save the changes:



Click YES

The screenshot shows the 'Controller Services' tab of the NiFi interface. It displays a table of services with columns: Name, Type, Bundle, State, and Scope. The table includes rows for AvroSchemaRegistryWeblogs, GrokReaderWeblogs, and JsonRecordSetWriterWeblogs. The 'JsonRecordSetWriterWeblogs' row has a gear icon in its scope column, which is highlighted in yellow.

Name	Type	Bundle	State	Scope
AvroSchemaRegistryWeblogs	AvroSchemaRegistry 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-registry-nar	Disabled	NiFi Flow
GrokReaderWeblogs	GrokReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Invalid	NiFi Flow
JsonRecordSetWriterWeblogs	JsonRecordSetWriter 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Disabled	NiFi Flow

click the gear icon for the JsonRecordSetWriterWeblogs

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field +

Property	Value
Schema Write Strategy	Do Not Write Schema
Schema Cache	No value set
Schema Access Strategy	Inherit Record Schema
Schema Registry	No value set
Schema Name	\$(schema.name)
Schema Version	No value set
Schema Branch	No value set
Schema Text	\$(avro.schema)
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set
Pretty Print JSON	false
Suppress Null Values	Never Suppress
Output Grouping	Array

CANCEL APPLY

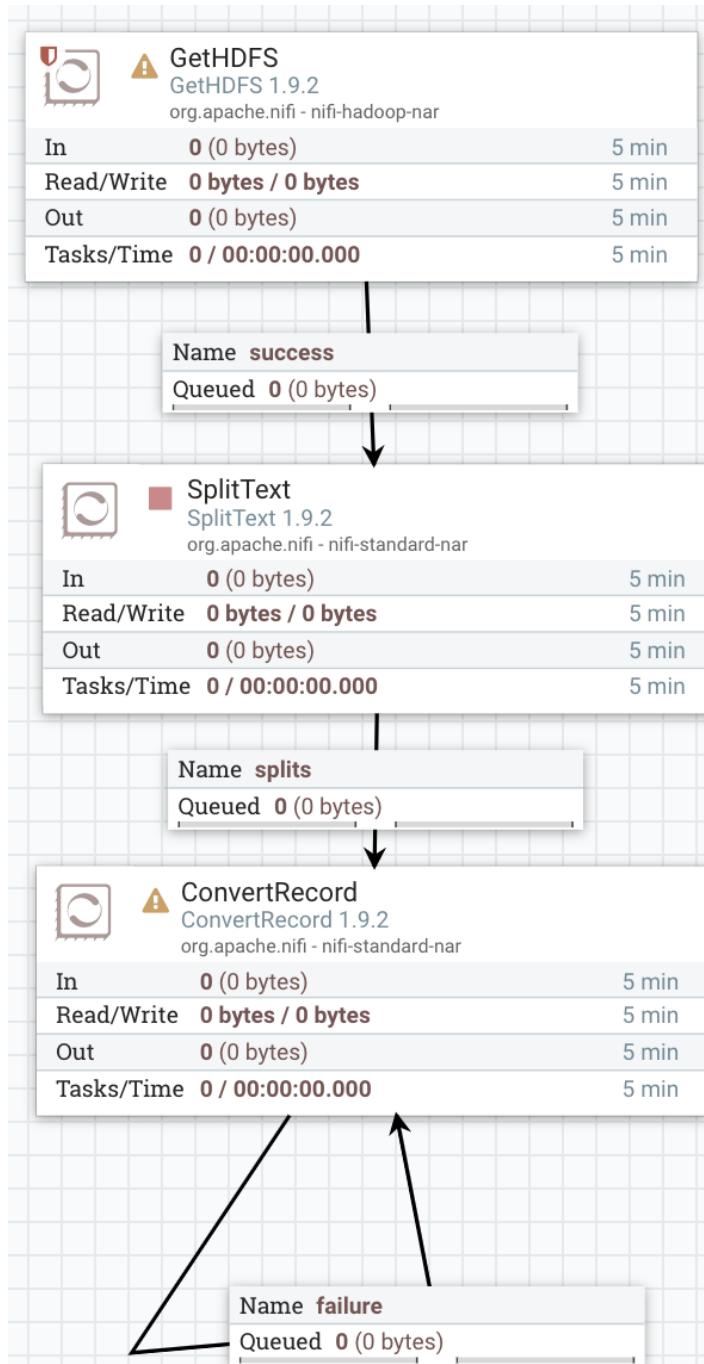
accept the default and click **APPLY**

Go back to the **ConvertRecord** and drag & drop a line from the **ConvertRecord** back to itself and check *failure*:

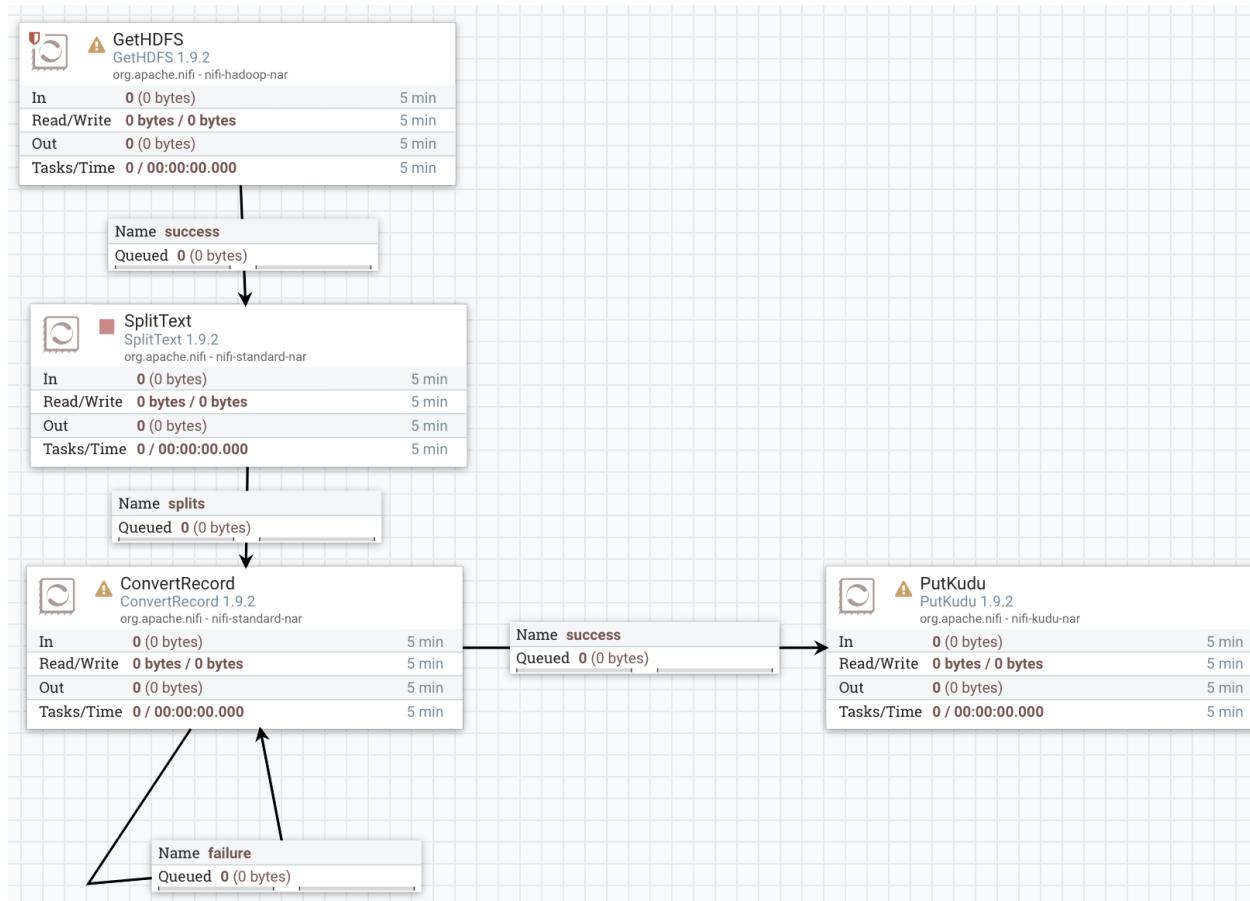
Create Connection

DETAILS SETTINGS

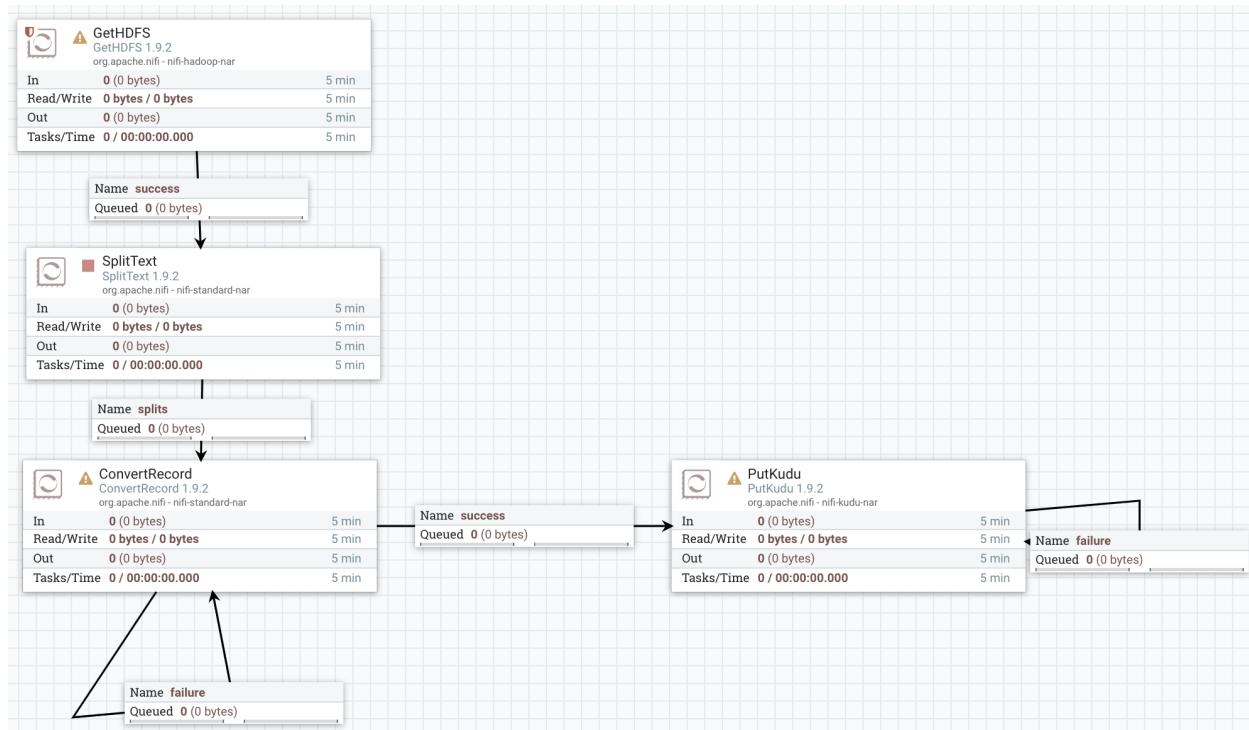
From Processor ConvertRecord ConvertRecord	To Processor ConvertRecord ConvertRecord
Within Group NiFi Flow	Within Group NiFi Flow
For Relationships <input checked="" type="checkbox"/> failure <input type="checkbox"/> success	



Then add the **PutKudu** processor and link it to the **ConvertRecord**:



and also for **PutKudu** drag & drop a connection back to itself and check **failure**:



Double click on **PutKudu** and go to the **SETTINGS** tab and automatically terminate in case of success:

### Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Name PutKudu	<input checked="" type="checkbox"/> Enabled	Automatically Terminate Relationships <small>?</small>	
Id 5ceaadbc-0c3b-33d7-040a-b4b4f4f3ccb9	<input type="checkbox"/> failure A FlowFile is routed to this relationship if it cannot be sent to Kudu		
Type PutKudu 1.9.2	<input checked="" type="checkbox"/> success A FlowFile is routed to this relationship after it has been successfully stored in Kudu		
Bundle org.apache.nifi - nifi-kudu-nar			
Penalty Duration <small>?</small> 30 sec	Yield Duration <small>?</small> 1 sec		
Bulletin Level <small>?</small> WARN			
<small>CANCEL</small> <small>APPLY</small>			

Then go to the **PROPERTIES** tab and set these properties:

**Kudu Masters:** <address of your Master node>:7051 (in my case flamb2-mn0.westeurope.cloudapp.azure.com:7051)

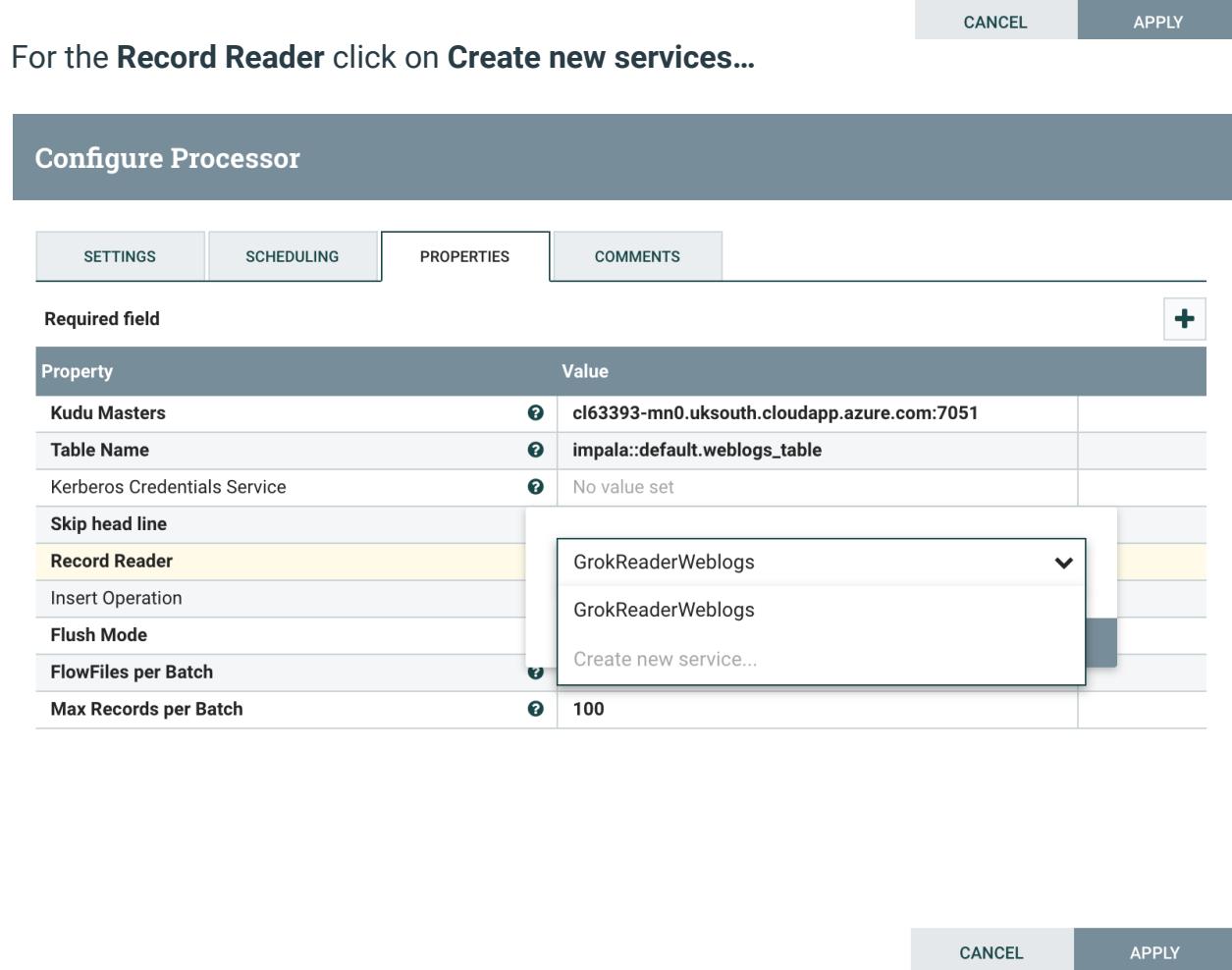
**Table Name:** impala::default.weblogs\_table

**Insert Operation:** UPSERT

### Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS																				
Required field																							
<table border="1"> <thead> <tr> <th>Property</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Kudu Masters</td> <td>cl62843-mn0.ukwest.cloudapp.azure.com:7051</td> </tr> <tr> <td>Table Name</td> <td>impala::default.weblogs_table</td> </tr> <tr> <td>Kerberos Credentials Service</td> <td>No value set</td> </tr> <tr> <td>Skip head line</td> <td>false</td> </tr> <tr> <td>Record Reader</td> <td>JsonTreeReaderWeblogs</td> </tr> <tr> <td>Insert Operation</td> <td>UPSERT</td> </tr> <tr> <td>Flush Mode</td> <td>AUTO_FLUSH_BACKGROUND</td> </tr> <tr> <td>FlowFiles per Batch</td> <td>1</td> </tr> <tr> <td>Max Records per Batch</td> <td>100</td> </tr> </tbody> </table>				Property	Value	Kudu Masters	cl62843-mn0.ukwest.cloudapp.azure.com:7051	Table Name	impala::default.weblogs_table	Kerberos Credentials Service	No value set	Skip head line	false	Record Reader	JsonTreeReaderWeblogs	Insert Operation	UPSERT	Flush Mode	AUTO_FLUSH_BACKGROUND	FlowFiles per Batch	1	Max Records per Batch	100
Property	Value																						
Kudu Masters	cl62843-mn0.ukwest.cloudapp.azure.com:7051																						
Table Name	impala::default.weblogs_table																						
Kerberos Credentials Service	No value set																						
Skip head line	false																						
Record Reader	JsonTreeReaderWeblogs																						
Insert Operation	UPSERT																						
Flush Mode	AUTO_FLUSH_BACKGROUND																						
FlowFiles per Batch	1																						
Max Records per Batch	100																						

For the **Record Reader** click on **Create new services...**



The screenshot shows the 'Configure Processor' interface with the 'PROPERTIES' tab selected. There are four tabs at the top: SETTINGS, SCHEDULING, PROPERTIES, and COMMENTS. Below the tabs is a section titled 'Required field' with a plus sign icon. A table lists properties and their values:

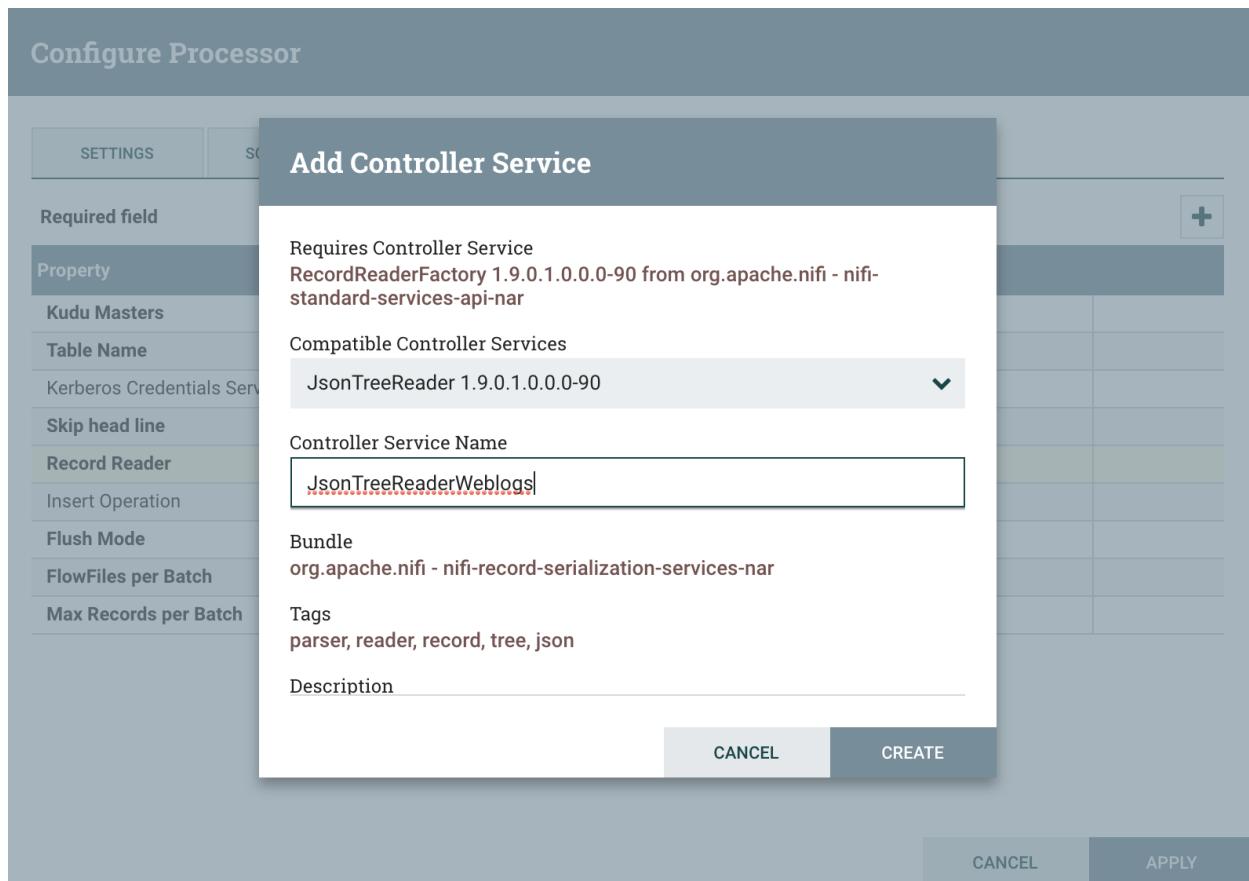
Property	Value
Kudu Masters	cl63393-mn0.ualtouth.cloudapp.azure.com:7051
Table Name	impala::default.weblogs_table
Kerberos Credentials Service	No value set

Below the table are several configuration options:

- Skip head line
- Record Reader** (highlighted in yellow)
- Insert Operation
- Flush Mode
- FlowFiles per Batch
- Max Records per Batch (value: 100)

A dropdown menu is open over the 'Record Reader' field, showing 'GrokReaderWeblogs' and 'Create new service...'. The 'Create new service...' option is highlighted.

In the **Compatible Controller Services** select **JsonTreeReader** and give it a name  
(JsonTreeReaderWeblogs)



**CREATE**

Configure Processor

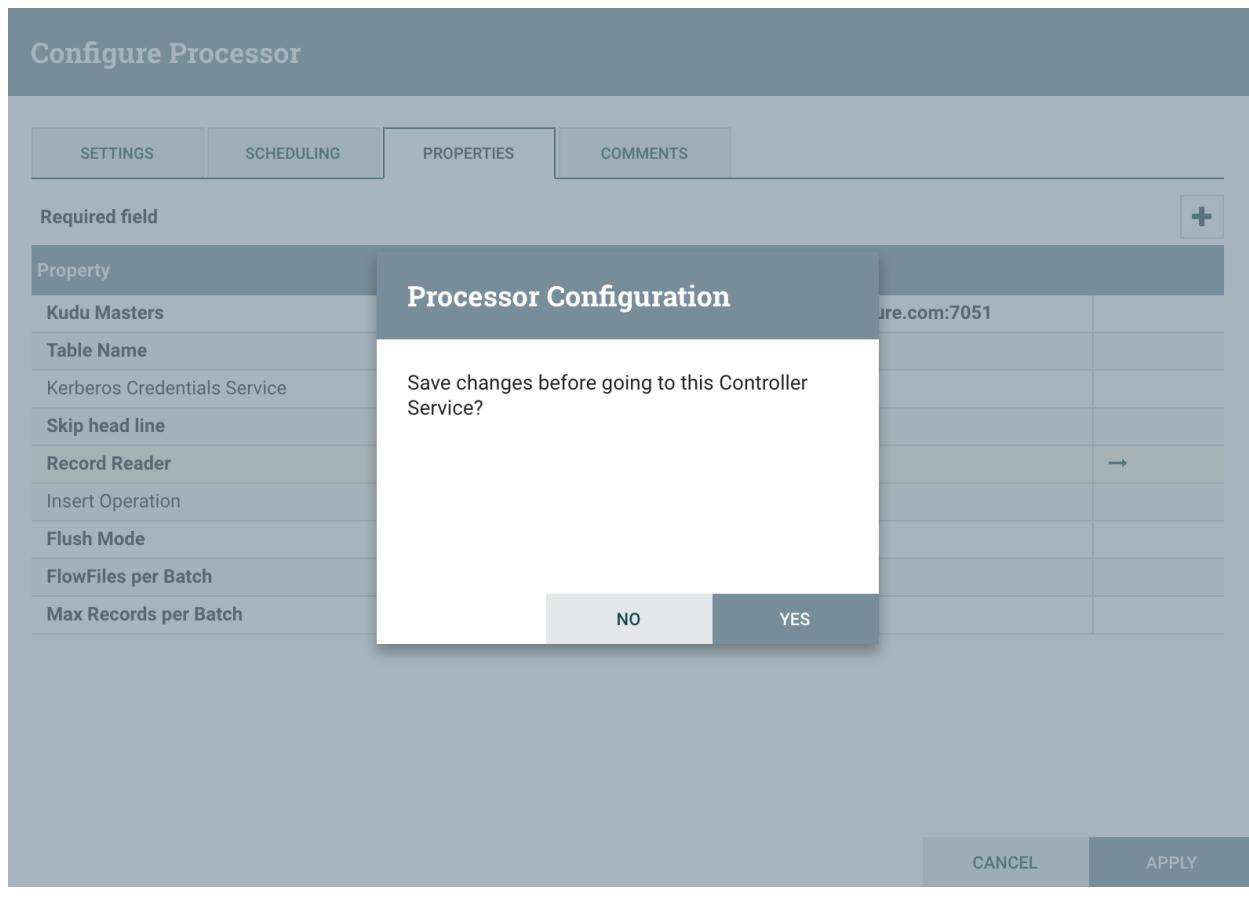
SETTINGS   SCHEDULING   PROPERTIES   COMMENTS

Required field

Property	Value	
Kudu Masters	cl63393-mn0.uksouth.cloudapp.azure.com:7051	
Table Name	impala::default.weblogs_table	
Kerberos Credentials Service	No value set	
Skip head line	false	
Record Reader	JsonTreeReaderWeblogs	→
Insert Operation	UPSERT	
Flush Mode	AUTO_FLUSH_BACKGROUND	
FlowFiles per Batch	1	
Max Records per Batch	100	

CANCEL   APPLY

arrow and save



Controller Services					
Name	Type	Bundle	State	Scope	
AvroSchemaRegistryWeblogs	AvroSchemaRegistry 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-registry-nar	✗ Disabled	NiFi Flow	
GrokReaderWeblogs	GrokReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	⚠ Invalid	NiFi Flow	
JsonRecordSetWriterWeblogs	JsonRecordSetWriter 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	✗ Disabled	NiFi Flow	
JsonTreeReaderWeblogs	JsonTreeReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	✗ Disabled	NiFi Flow	

then configure it as follows

Configure Controller Service

SETTINGS PROPERTIES COMMENTS

Required field

Property	Value
Schema Access Strategy	Use 'Schema Name' Property
Schema Registry	AvroSchemaRegistryWeblogs
Schema Name	weblogs
Schema Version	No value set
Schema Branch	No value set
Schema Text	\$(avro.schema}
Schema Inference Cache	No value set
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set

+ CANCEL APPLY

where

**Schema Registry** is **AvroSchemaRegistryWeblogs** (the one also used before)

**Schema Name** is **weblogs**

Click **APPLY** and go back to the **PutKudu** processor

## Configure Processor

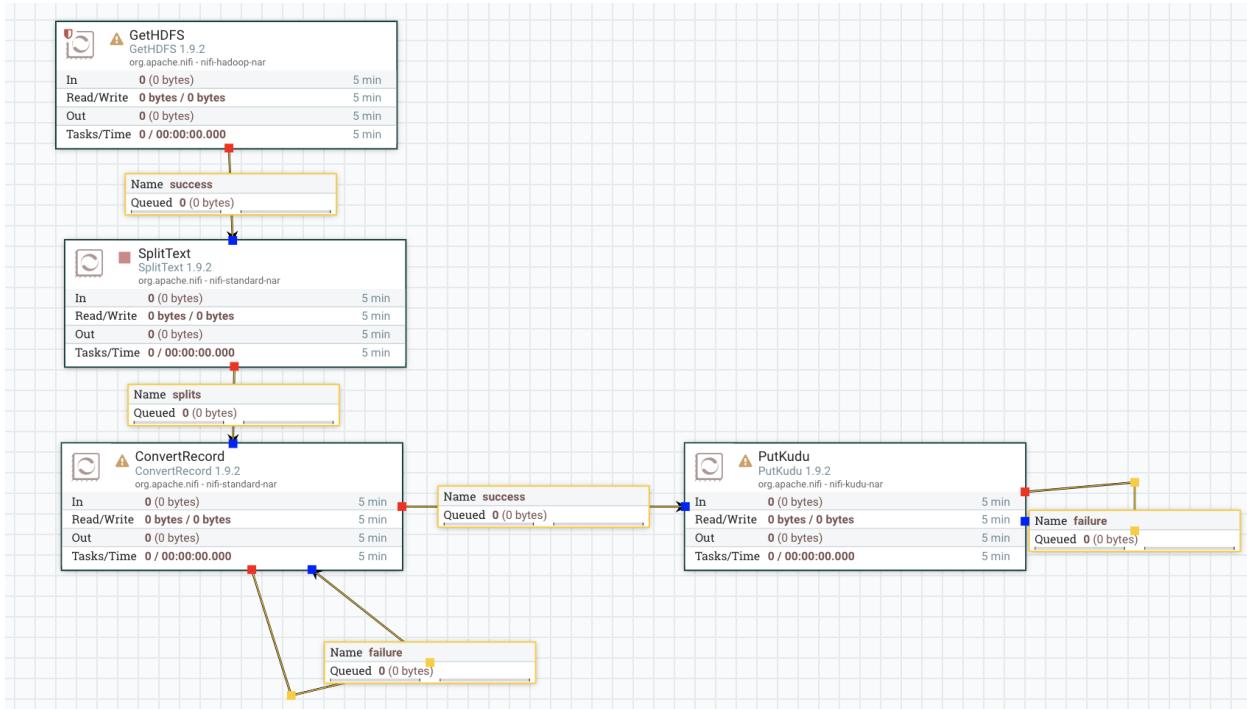
SETTINGS   SCHEDULING   PROPERTIES   COMMENTS

Required field +

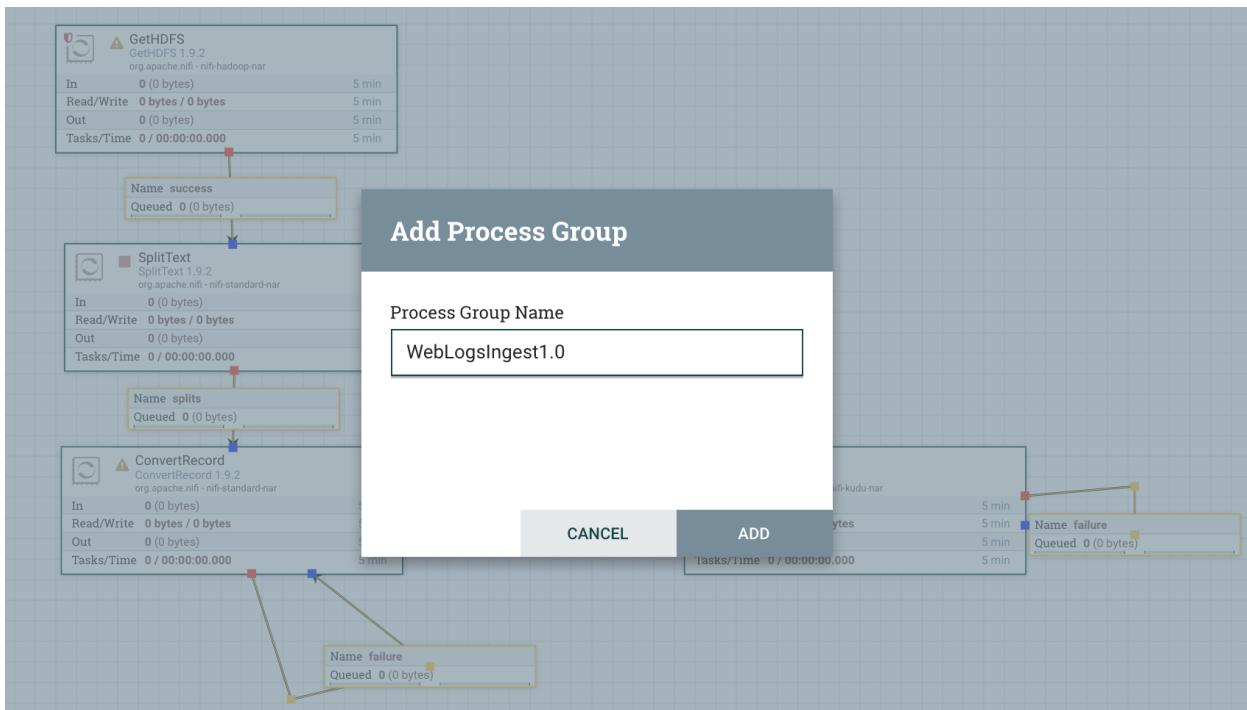
Property	Value
Kudu Masters	cl62843-mn0.ukwest.cloudapp.azure.com:7051
Table Name	impala::default.weblogs_table
Kerberos Credentials Service	No value set
Skip head line	false
Record Reader	JsonTreeReaderWeblogs
Insert Operation	UPSERT
Flush Mode	AUTO_FLUSH_BACKGROUND
FlowFiles per Batch	1
Max Records per Batch	100

CANCEL   APPLY

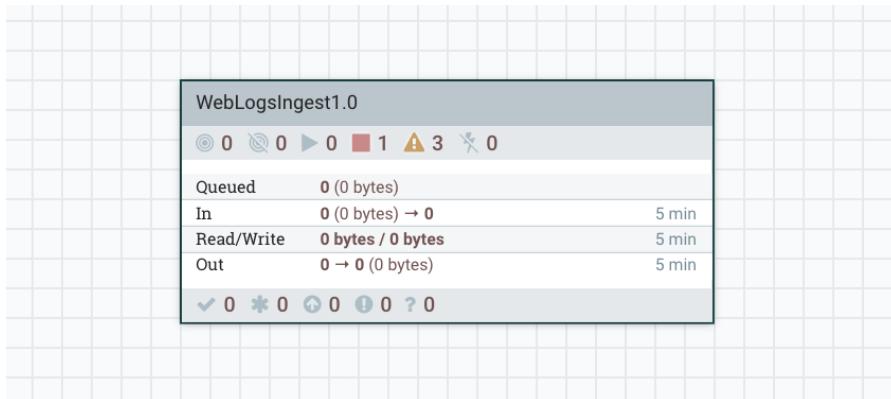
Now that the flow is finished, let's create a Group. With the mouse select a point in the canvas, press **SHIFT** and hold down the mouse and select your flow:



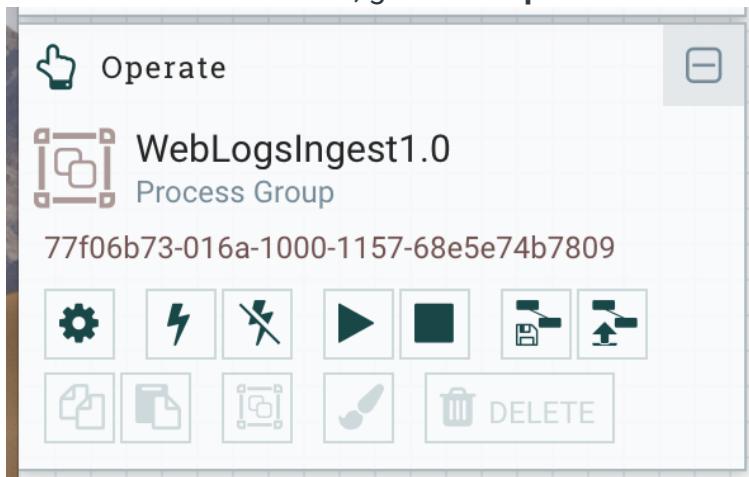
Then right click and click **Group** and give it a name:



click **ADD**



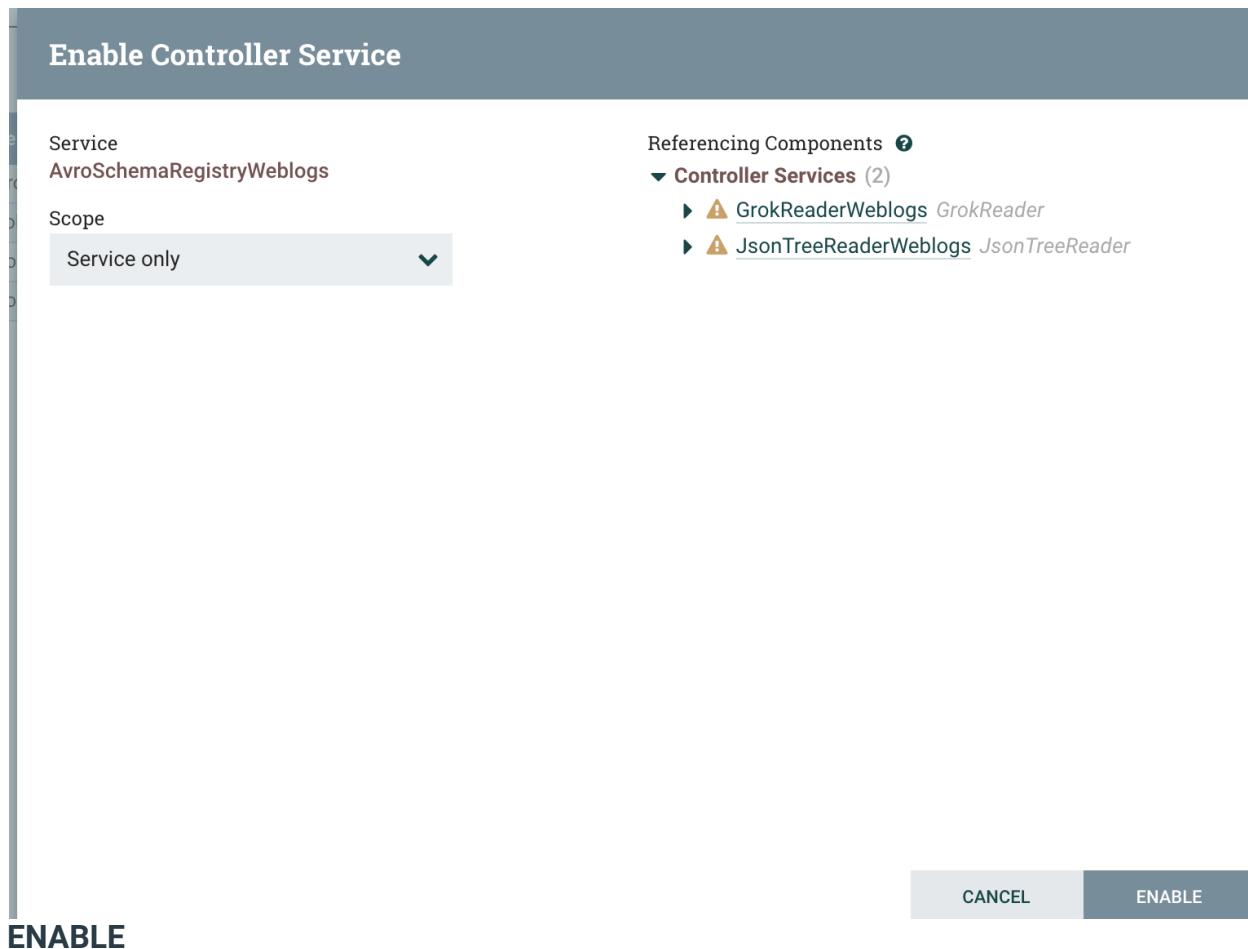
Now we can run the flow, go to the **Operate** box:



Click on the gear icon (configuration), go to the **CONTROLLER SERVICES** tab

Name	Type	Bundle	State	Scope
AvroSchemaRegistryWeblogs	AvroSchemaRegistry 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-registry-nar	✗ Disabled	NiFi Flow
GrokReaderWeblogs	GrokReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	⚠ Invalid	NiFi Flow
JsonRecordSetWriterWeblogs	JsonRecordSetWriter 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	✗ Disabled	NiFi Flow
JsonTreeReaderWeblogs	JsonTreeReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	⚠ Invalid	NiFi Flow

and Enable those that we have created by clicking the lightning icon.



Service  
AvroSchemaRegistryWeblogs

Steps To Enable AvroSchemaRegistryWeblogs  
Enabling this controller service

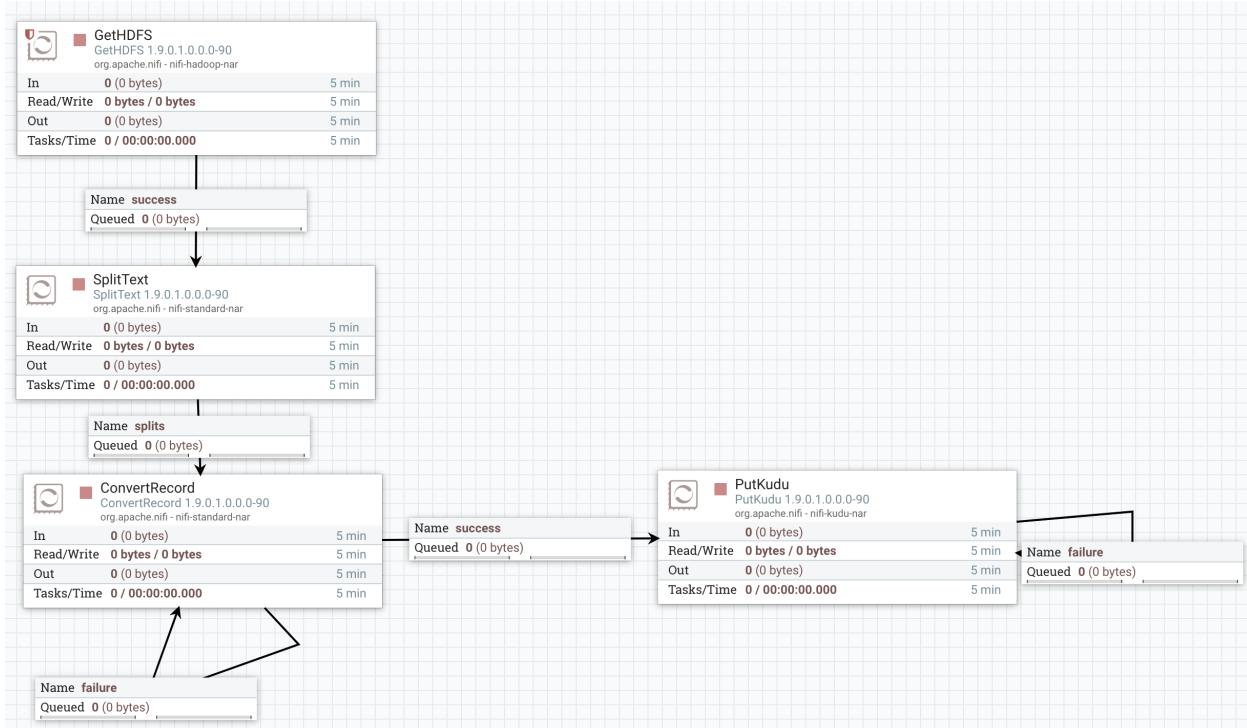
Referencing Components ?  
▼ Controller Services (2)  
▶ GrokReaderWeblogs GrokReader  
▶ JsonTreeReaderWeblogs JsonTreeReader

CLOSE

CLOSE and enable all the others

Name	Type	Bundle	State	Scope
AvroSchemaRegistryWeblogs	AvroSchemaRegistry 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-registry-nar	Enabled	NiFi Flow
GrokReaderWeblogs	GrokReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Enabling	NiFi Flow
JsonRecordSetWriterWeblogs	JsonRecordSetWriter 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Enabling	NiFi Flow
JsonTreeReaderWeblogs	JsonTreeReader 1.9.0.1.0.0.0-90	org.apache.nifi - nifi-record-serialization-service...	Enabled	NiFi Flow

Go back to the flow and now you should see that all the processors are the red square icon and no more the warning icon

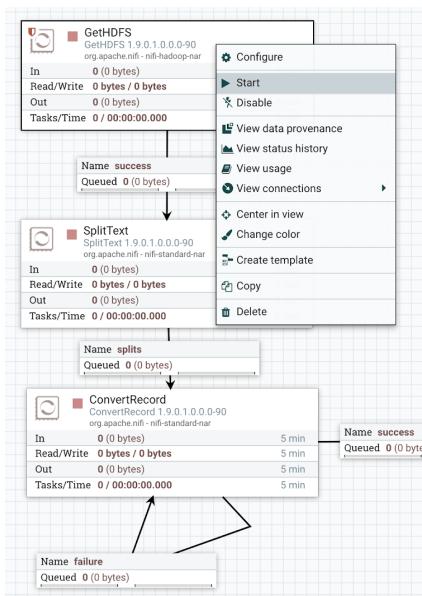


Now our flow is ready to run....

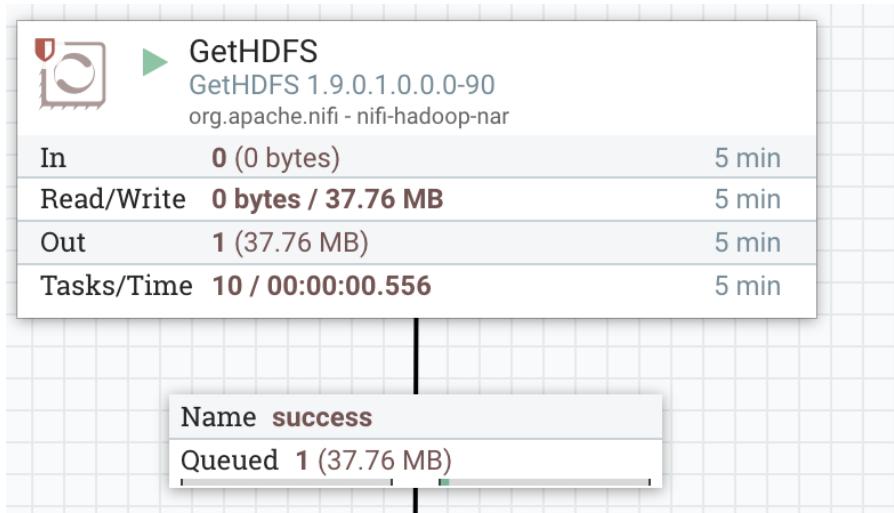
## Run the Flow

We are going to run the flow step by step.

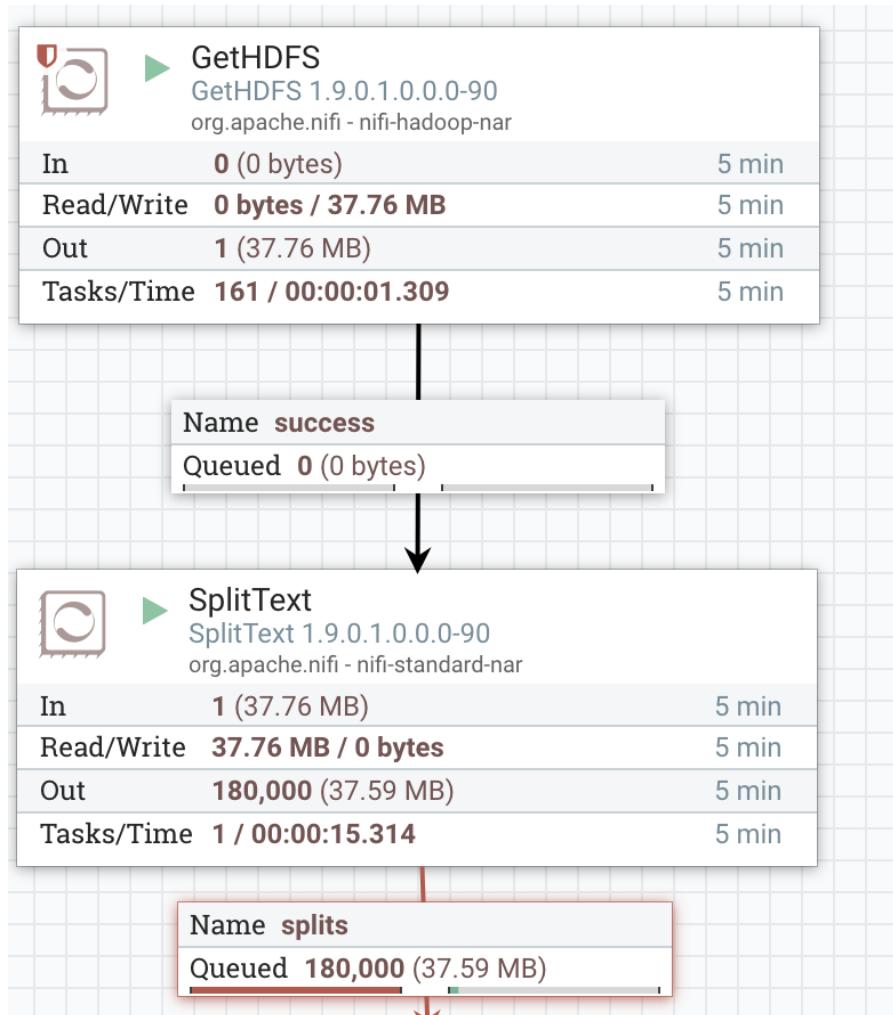
Select the **GetHDFS**, right click and Start



Refresh and then we should have the file in the queue

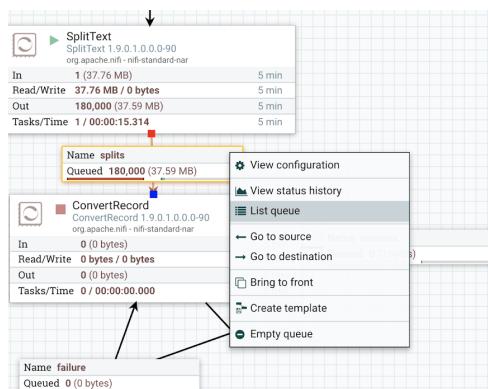


Start the SplitText



The red line in the queue is normal, it just indicates that the “**back pressure**” control kicked in.

Right click on the queue and **List queue**



# Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

splits

Displaying 100 of 180,000 (37.59 MB)

The source of this queue is currently running. This listing may no longer be accurate.

Position	UUID	Filename	File Size	Queued Duration	Lineage Duration	Penalized	Node
1	0cb4872-ebda-4766-97b9-341dbb...	web.log (1).2	193.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
2	b982d157-7736-408c-bd6c-bb17765...	web.log (1).2	301.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
3	9cefef23-081f-45bb-a473-3ff61d3...	web.log (1).2	217.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
4	d0549e2a-8022-435a-81a3-7aac1f1...	web.log (1).2	298.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
5	a37023bf-bdf1-4862-ae9e-cd354c8...	web.log (1).2	160.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
6	d95ac662-8c0a-4122-a9fa-030ad19...	web.log (1).2	196.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
7	a6827b44-8783-d481-9173-7a6c74c...	web.log (1).2	208.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
8	b78d43d9-e524-47a8-a83e-babca3d8634f	web.log (1).2	238.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
9	80aa3c2c-03e2-493f-9800-e0190c...	web.log (1).2	291.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
10	720d7588-1057-490c-91eb-46d056...	web.log (1).2	274.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
11	3bcaffdd-ddbb-4059-90ff-8957675b...	web.log (1).2	193.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
12	f7e514ec-a998-4c07-a9a8-e8c5e9...	web.log (1).2	217.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
13	04d53342-4a54-4cfe-94d9-12cf021...	web.log (1).2	196.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
14	afc3759a-b9f7-4b66-a14f-cca55118...	web.log (1).2	186.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
15	b53c24ee-13d8-4af8-874a-bb7d9f5...	web.log (1).2	186.00 bytes	00:02:34.005	00:04:54.117	No	cl63393-dn0.uksouth.cloudapp.azure.com:8080
...	...	...	...	...	...	...	...
16	da27917a-1a6e-4777-8a7a-4a...	web.log (1).2	300.00 bytes	00:02:34.005	00:04:54.117	...	...

Click on the info icon on an item

File Size	Queued Duration	Lineage Duration
274.00 bytes	00:02:34.005	00:04:54.117

**FlowFile**

**FlowFile Details**

**UUID**  
b982d157-7736-408c-bd6c-bb17765fb83

**Filename**  
web.log (1).2

**File Size**  
301 bytes

**Queue Position**  
No value set

**Queued Duration**  
00:03:18.213

**Lineage Duration**  
00:05:38.325

**Penalized**  
No

**Node Address**  
cl63393-dn0.uksouth.cloudapp.azure.com:8080

**Content Claim**

**Container**  
default

**Section**  
1

**Identifier**  
1556877064746-1

**Offset**  
194

**Size**  
301 bytes

DOWNLOAD
 VIEW

OK

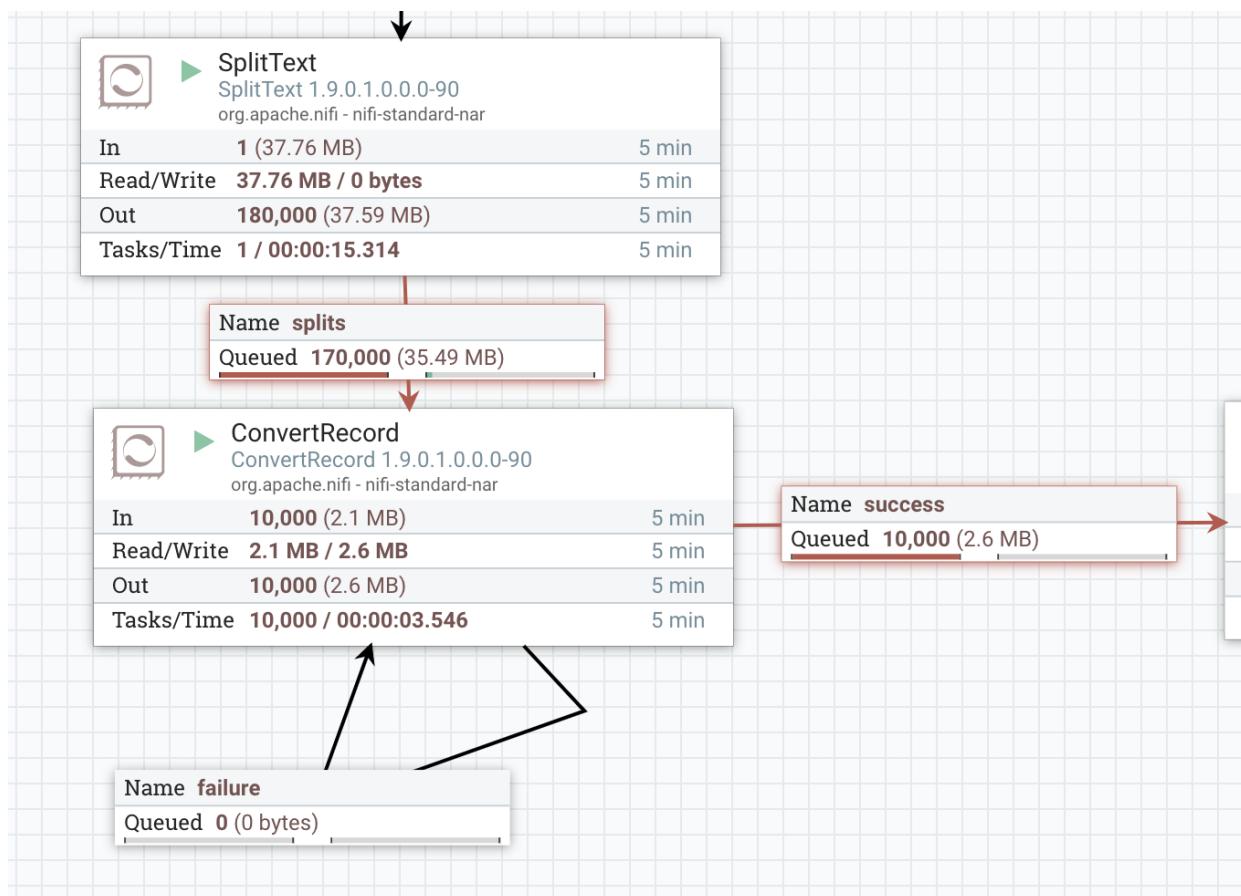
and then **VIEW**

```
View as: original ▾
1 162.235.161.200 - - [14/Jun/2014:10:30:13 -0400] "GET /department/apparel/category/featured%20shops/product/adidas%20Kids%20ORG%20III%20Mid%20Football%20Cleat HTTP/1.1" 200 1175 "-" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.111 Safari/537.36"
```

Filename: web.log (1).2  
Content-Type: text/plain

this is one line of the web logs file.

## Start the ConvertRecord



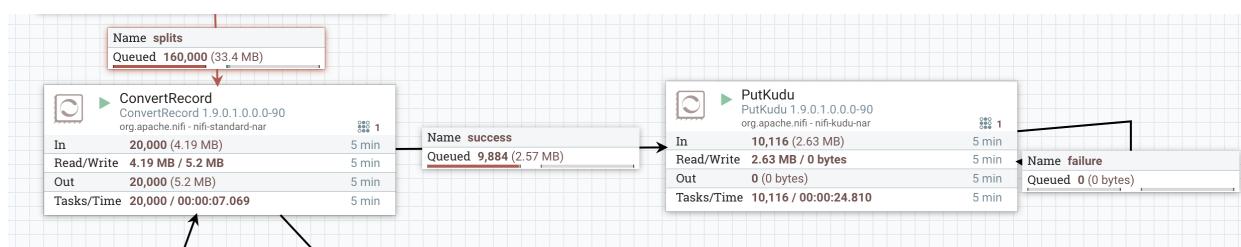
and check on item of the queue that connect the PutKudu

View as: formatted

```

1 [ {
2   "ip" : "79.133.215.123",
3   "log_date" : "14/Jun/2014:10:30:13 -0400",
4   "log_method" : "GET",
5   "url" : "/home",
6   "http_version" : "1.1",
7   "code1" : "200",
8   "code2" : "1671",
9   "user_agent" : "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"
10 } ]
  
```

Then run the PutKudu



## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

As you can see the queues are getting empty...  
In the meantime go to impala and check the weblogs\_table.

**select count(\*) from weblogs\_table;**

The screenshot shows the Impala Query Editor interface. At the top, there is a code editor with two lines of SQL: '16' and '17 select count(\*) from weblogs\_table;'. Below the code editor is a toolbar with a play button and a save icon. The main area displays the results of the query: 'Query f14cc393e9a1653b:5f33bfed00000000 100% Complete (16 out of 16)' followed by a green link 'f14cc393e9a1653b:5f33bfed00000000'. Below this, there are tabs for 'Query History', 'Saved Queries', and 'Results (1)'. The 'Results (1)' tab is selected, showing a single row with the value '59002'. The bottom part of the interface shows a table with columns 'ip', 'log\_date', 'log\_method', and 'url', containing 10 rows of log data.

	ip	log_date	log_method	url
1	1.111.125.242	14/Jun/2014:22:15:24 -0400	GET	/home
2	1.111.125.242	14/Jun/2014:22:15:49 -0400	GET	/department/footwear
3	1.111.125.242	14/Jun/2014:22:16:12 -0400	GET	/department/fan%20shop
4	1.111.125.242	14/Jun/2014:22:17:46 -0400	GET	/department/footwear
5	1.111.125.242	14/Jun/2014:22:18:02 -0400	GET	/department/footwear/category/cardio%20equipment/product/Nike%20Men's%
6	1.111.125.242	14/Jun/2014:22:19:02 -0400	GET	/department/golf/category/girls%20apparel
7	1.111.125.242	14/Jun/2014:22:19:36 -0400	GET	/department/fitness
8	1.111.125.242	14/Jun/2014:22:19:43 -0400	GET	/home
9	1.18.156.107	14/Jun/2014:21:51:35 -0400	GET	/department/fitness/category/lacrosse

**select \* from weblogs\_table limit 10;**

The screenshot shows the Impala Query Editor interface. At the top, there is a code editor with two lines of SQL: '18' and '19 select \* from weblogs\_table limit 10;'. Below the code editor is a toolbar with a play button and a save icon. The main area displays the results of the query: 'Query 80436b72fbfc322c:1c9ab9d700000000: 6% Complete (1 out of 16)' followed by a green link '80436b72fbfc322c:1c9ab9d700000000'. Below this, there are tabs for 'Query History', 'Saved Queries', and 'Results (10)'. The 'Results (10)' tab is selected, showing a table with columns 'ip', 'log\_date', 'log\_method', and 'url', containing 10 rows of log data.

	ip	log_date	log_method	url
1	1.111.125.242	14/Jun/2014:22:15:24 -0400	GET	/home
2	1.111.125.242	14/Jun/2014:22:15:49 -0400	GET	/department/footwear
3	1.111.125.242	14/Jun/2014:22:16:12 -0400	GET	/department/fan%20shop
4	1.111.125.242	14/Jun/2014:22:17:46 -0400	GET	/department/footwear
5	1.111.125.242	14/Jun/2014:22:18:02 -0400	GET	/department/footwear/category/cardio%20equipment/product/Nike%20Men's%
6	1.111.125.242	14/Jun/2014:22:19:02 -0400	GET	/department/golf/category/girls%20apparel
7	1.111.125.242	14/Jun/2014:22:19:36 -0400	GET	/department/fitness
8	1.111.125.242	14/Jun/2014:22:19:43 -0400	GET	/home
9	1.18.156.107	14/Jun/2014:21:51:35 -0400	GET	/department/fitness/category/lacrosse

Try some more queries:

**select count(\*),url from weblogs\_table  
where url like '%\product\%'  
group by url order by count(\*) desc;**

## Cloudera Tech 2019 - developed by Filippo Lambiente - EMEA PSE

The screenshot shows a SQL query results page. At the top, there is a code editor with the following SQL query:

```
21
22 select count(*),url from weblogs_table
23 where url like '%product%'
24 group by url order by count(*) desc;
25
```

Below the code editor, the status bar indicates "Query 514658d210fb7039:6f2200a300000000 100% Complete (16 out of 16)". To the right of the status bar is a link: "514658d210fb7039:6f2200a300000000".

The main area displays a table with two columns: "count(\*)" and "url". The table contains 11 rows of data, with the first 10 rows explicitly listed:

	count(*)	url
1	1379	/department/apparel/category/cleats/product/Perfect%20Fitness%20Perfect%20Rip%20Deck
2	1289	/department/apparel/category/featured%20shops/product/adidas%20Kids%20RG%20III%20Mid%20Football%20Cleat
3	1283	/department/apparel/category/men's%20footwear/product/Nike%20Men's%20CJ%20Elite%20TD%20Football%20Cleat
4	1259	/department/golf/category/women's%20apparel/product/Nike%20Men's%20Dri-FIT%20Victory%20Golf%20Polo
5	800	/department/fan%20shop/category/water%20sports/product/Pelican%20Sunstream%20100%20Kayak
6	762	/department/fan%20shop/category/indoor%20outdoor%20games/product/O'Brien%20Men's%20Neoprene%20Life%20Vest
7	739	/department/footwear/category/cardio%20equipment/product/Nike%20Men's%20Free%205.0+%20Running%20Shoe
8	701	/department/footwear/category/fitness%20accessories/product/Under%20Armour%20Hustle%20Storm%20Medium%20Duffle%20Tote
9	696	/department/golf/category/shop%20by%20sport/product/Columbia%20Men's%20PFG%20Anchor%20Tough%20T-Shirt
10	634	/department/footwear/category/as%20seen%20on%20tv!/product/Nike%20Men's%20Free%20TR%205.0%20TB%20Training%20Shirt

At the following link you can download the already built flow:

<https://cloudera.box.com/s/14hwbdlhqdb8j8cv33dwjnmes9qvicz>

Now we will do further analysis – lets try and find out which customer and which IP address had the most click throughs.

Can you pick the IP address that have the highest number of Click Through Rates. Try and create a SQL Query to write this

```
select ip, count(*) CTR_click_through from weblogs_table
group by ip
order by CTR_click_through desc;
```

The screenshot shows the Cloudera Impala Query Editor interface. At the top, there is a code editor with the following SQL query:

```
25  
26  
27  
28 select ip, count(*) CTR_click_through from weblogs_table  
29 group by ip  
30 order by CTR_click_through desc;  
31
```

Below the code editor, a message indicates the query is complete: "Query cb418b61e586722b:e8cf24f800000000 100% Complete (16 out of 16)". To the right of the message is a link: "cb418b61e586722b:e8cf24f800000000".

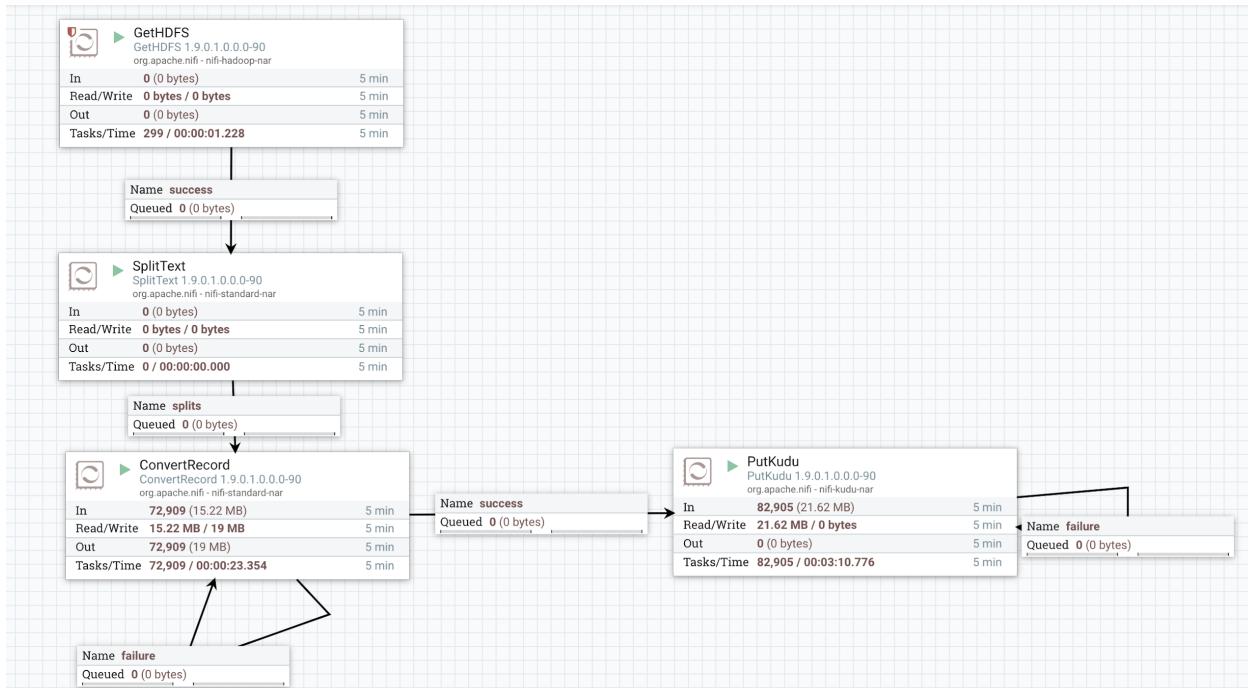
At the bottom, there are tabs for "Query History" and "Saved Queries", and a "Results (1.122)" tab which is currently selected. The results table has two columns: "ip" and "ctr\_click\_through". The data is as follows:

	ip	ctr_click_through
1	17.40.8.46	78
2	105.52.11.168	77
3	77.137.114.147	77
4	17.224.101.119	76
5	138.21.216.113	75
6	120.188.211.38	75
7	95.24.232.236	75

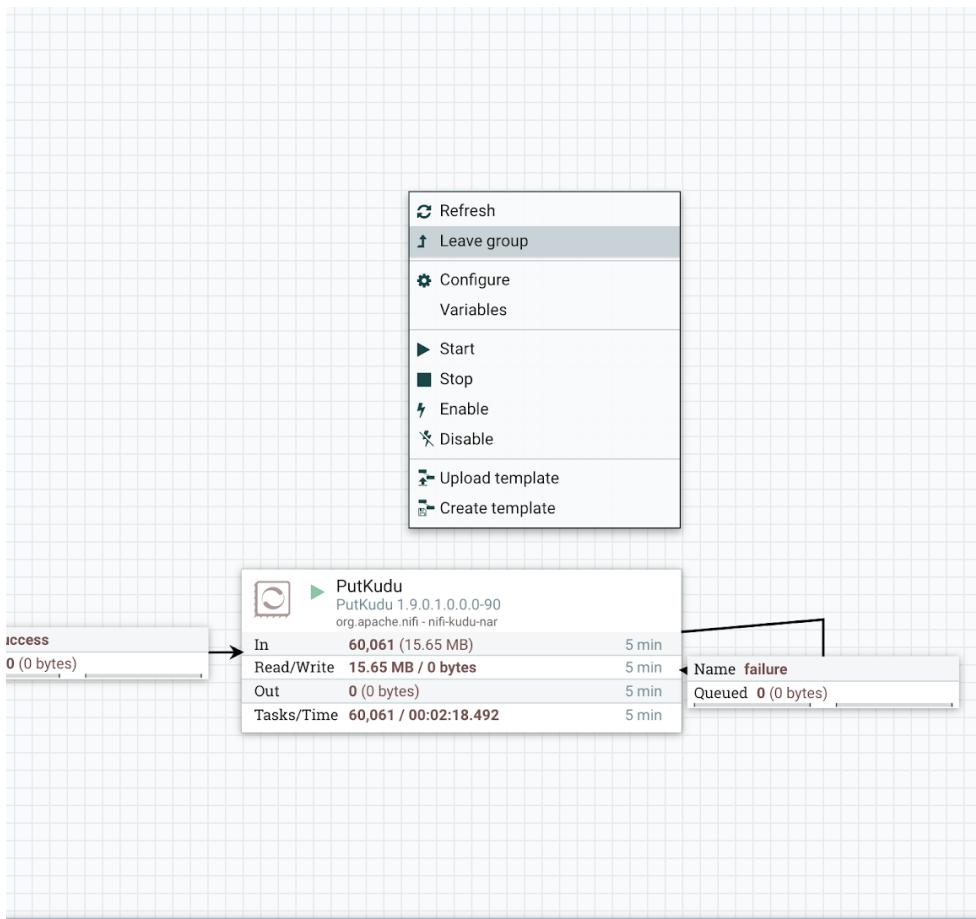
We can do further analysis on where these IP address Geolocation using a pySpark job and finding out which states have the highest number of click throughs on our retail website.

We could even tie IP address to customer based on customer login details and create a Customer360 of our retail customer profile and shopping habits.

If you go back to NiFi then you should see that all the messages have been consumed:

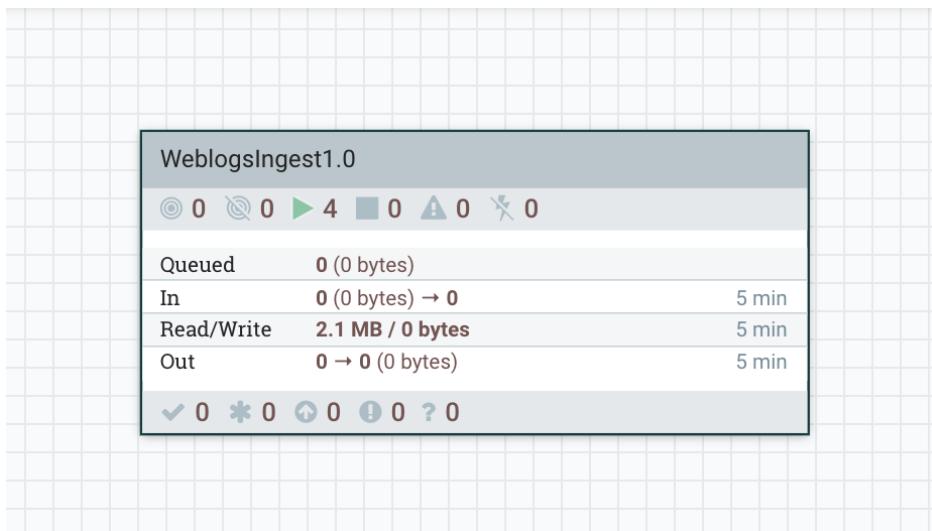


Now you can stop the flow, click on a point in the canvas

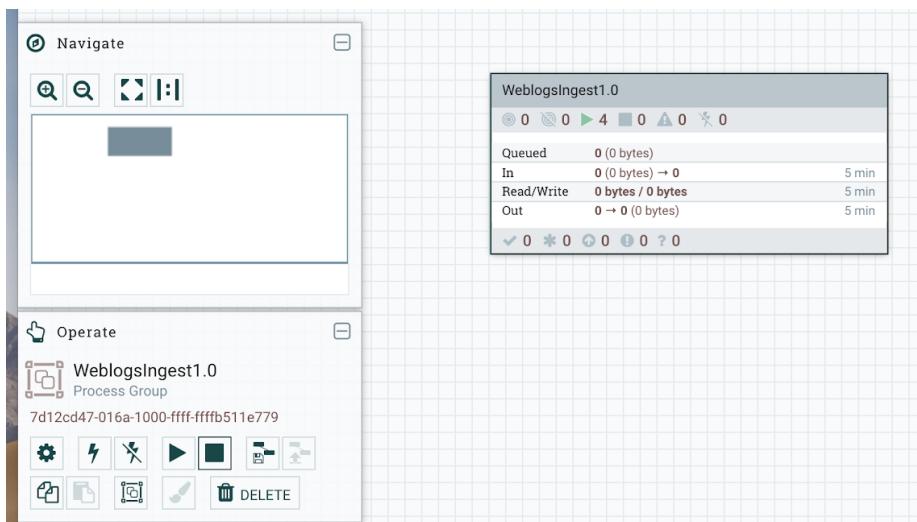


**Leave group**

then select the WeblogsIngest1.0



and click **Stop** in the Operate box



and our flow is stopped

